VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

USING MATRIX DECOMPOSITION TO IMPROVE CONCEPT LATTICE REDUCTION

Hussam Mahfood Dahwa Abdulla

Field of study: Computer Science and Applied Mathematics
Supervisor: Prof. RNDr. Václav Snášel, CSc.

Ostrava, 2012

To my loving wife, my dear daughters and my dear parents.

Preface

The large volume of data from the large-scale computing platforms for high-fidelity design and simulations and instrumentation for gathering scientific, as well as, business data, and the huge information in the web, can be problematic if we want to compute all concepts from huge incidence matrix. High complexity of formal concept analysis algorithms and lattice construction algorithms are the main problems today. If we want to compute all concepts from huge incidence matrix, complexity plays a great role. In some cases, we do not need to compute all concepts, but only some of them. This thesis proposed minimizing incidence matrix using matrix decompositions (Singular Value Decomposition SVD and non-negative matrix factorization NMF). Modified matrix has lower dimensions and acts as input for some known algorithms for lattice construction. In this work, I would like to describe methods for matrix decompositions and describe their influence on the concept lattice.

Acknowledgments

I would like to express my special thanks and gratitude to my supervisor, prof. RNDr. Václav Snášel CSc. for all his support during my study. I would like also to thank my wife for standing by me. I would to extend my thanks to all my friends who helped me to finish this work, especially, Hussein Soori, Fathi Ali and Asim Ali.

Contents

1 Introduction **1**
- 1.1 Goal of research . . . 2
- 1.2 History of Formal Concept Analysis . . . 3
  - 1.2.1 The relationship of FCA notions to similar notions in other fields . . . 3
  - 1.2.2 FCA in Information Retrieva . . . 4
  - 1.2.3 FCA as a tool for knowledge representation and knowledge discovery . . . 5
  - 1.2.4 Applications of FCA in logic and AI . . . 7
- 1.3 Thesis outline . . . 8

2 Concept Lattices **10**
- 2.1 Basic notion of orders and lattices . . . 10
- 2.2 Context, concept, and concept lattice . . . 13
- 2.3 Algorithms . . . 16
  - 2.3.1 Computational space complexity of concept lattices . . . 16
  - 2.3.2 Construction of the set of concepts . . . 17
  - 2.3.3 Construction of concept lattices . . . 20
  - 2.3.4 Visualization . . . 23

3 Matrix decompositions **26**
- 3.1 Symmetry between objects and attributes . . . 28
- 3.2 Normalization . . . 28
- 3.3 Degenerate decompositions . . . 29
- 3.4 Correlation matrices . . . 29
- 3.5 Similarity and clustering . . . 30
  - 3.5.1 Geometric clustering . . . 30
  - 3.5.2 Decomposition-based clustering; similarity clustering . . . 30
  - 3.5.3 Graph-based clustering . . . 30
- 3.6 Finding local relationships . . . 31
- 3.7 Sparse representations . . . 31
- 3.8 Algorithms and complexity . . . 32

4 Singular Value Decomposition (SVD) **34**
- 4.1 SVD and Principal Component Analysis (PCA) . . . 35
- 4.2 Normalization . . . 36

4.3 Interpreting an SVD . . . 37
4.3.1 Factor interpretation . . . 37
4.3.2 Geometric interpretation . . . 39
4.3.3 Rotation and stretching . . . 39
4.3.4 Springs . . . 41
4.3.5 Component interpretation . . . 42
4.3.6 Graph interpretation . . . 42
4.4 Applying SVD . . . 43
4.4.1 Selecting factors, dimensions, components, and way stations 43
4.4.2 Selecting objects and/or attributes with special properties . 43
4.4.3 Denoising . . . 44
4.4.4 Similarity and clustering . . . 46
4.4.5 Finding local relationships . . . 47
4.5 Algorithm issues . . . 49
4.5.1 Algorithms and complexity . . . 49
4.5.2 Updating an SVD . . . 49

5 Non-Negative Matrix Factorization (NMF) 50
5.1 Factor interpretation . . . 52
5.2 Geometric interpretation . . . 53
5.3 Graph interpretation . . . 53
5.4 Applying an NNMF . . . 53
5.4.1 Selecting factors . . . 53
5.4.2 Denoising . . . 54
5.4.3 Similarity and clustering . . . 54
5.5 Algorithm issues . . . 54

6 Using Matrix decomposition in Concept Lattice 56
6.1 Computing concept lattices from the original data . . . 57
6.2 Computing SVD . . . 59
6.3 Computing concept lattices from data after using SVD . . . 62
6.4 Computing NMF . . . 67
6.5 Computing concept lattices from data after using NMF . . . 69
6.6 Behavior of the Concept Lattice Reduction to visualizing data after Using Matrix Decompositions . . . 74

7 Using Concept Lattice and Matrix decomposition in Search Results Clustering 76
7.1 Problem formalization and algorithm . . . 77
7.2 Using Singular Value Decomposition SVD . . . 79

8 Conclusions 85
8.1 Future work . . . 87

9 Author's publications 89

10 References **92**

List of Figures

2.1 Alternative line diagram 11
2.2 Line diagram of all possible ordered sets with three elements 11
2.3 The top and the bottom elements of the ordered set 12
2.4 Example of a product of two ordered sets 13
2.5 A context for objects 14
2.6 Concept lattice for the context of table 16

3.1 A basic matrix decomposition 26
3.2 Each element of A is expressed as a product of a row of 27

4.1 k-reduced singular value decomposition 36
4.2 The first two new axes when the data values are positive (top) and zero-centered (bottom) 38
4.3 The first two factors for a dataset ranking wines 39
4.4 One intuition about SVD: rotating and scaling the axes 40
4.5 Data appears 2-dimensional but can be seen to be 1-dimensional after rotation 41
4.6 The effect of noise on the dimensionality of a dataset 45

5.1 k-reduced Nonnegative Matrix Factorization (NMF) 52

6.1 Using SVD and NMF as improved factor to Concept Lattice reduction 56
6.2 Formal context from the original data 57
6.3 Nodes content in Concept Lattice 58
6.4 Concept lattice computed from simplified formal context (Figure 6.2) 59
6.5 k-rank for singular value decomposition 59
6.6 Simplified formal context after using SVD with k-rank 60
6.7 Representing data after combining the similar Objects 61
6.8 Groups content 61
6.9 Concept lattice computed from simplified formal context after using SVD with k-rank (figure 6.6) 62
6.10 Nodes content in concept lattice after using Singular Value decomposition (SVD) 63
6.11 The upper node in the original concept lattice before using SVD . . 63
6.12 The upper node in the concept lattice after using SVD 64
6.13 Nodes 2, 5 and 9 in the original concept lattice come under nod 1 in the new lattice after using SVD 64

6.14 Nodes 0 and 18 in the original concept lattice come under node 4 in the new lattice after using SVD 65
6.15 Node 3 is deleted. Nodes 4, 7 and 11 in the original concept lattice come under node 4 in the new lattice after using SVD 66
6.16 k-reduced Nonnegative Matrix Factorization (NMF) 67
6.17 Simplified formal context after using NMF with k-rank 68
6.18 representing data after combining the similar objects 68
6.19 Groups of content 69
6.20 Concept lattice computed from simplified formal context after using NMF with k-rank (figure 6.17) 69
6.21 Nodes content in Concept Lattice after using Nonnegative Matrix Factorization (NMF) 70
6.22 The upper node in the original concept lattice before using NMF . . 70
6.23 The upper node in the concept lattice after using NMF 71
6.24 Nodes 2, 5, 9 and 1 in the original concept lattice came under node 0 in the new lattice after using NMF 71
6.25 Nodes 0 and 18 in the original concept lattice came under node 8 in the new lattice after using NMF 72
6.26 combination of attributes 73
6.27 Nodes 3, 14, 19 and 20 in the original concept lattice came under node 9 in the new lattice 73
6.28 behavior of the Concept Lattice Reduction to visualizing data after Using Matrix Decompositions methods 75

7.1 Steps of the algorithm process 78
7.2 The concept lattice computed from the table 7.1 79
7.3 Data deducted from the result of Google search engine 80
7.4 Data after using SVD with k-rank 81
7.5 Representing data after combining the similar Objects 82
7.6 The concept lattice computed from the table 7.2 82
7.7 Relationship between value of k-rank and number of combination . 83
7.8 content of the groups 83
7.9 Map of the groups and their content 84

1 Introduction

The big progress in dealing with data such as storage and data transfer due to the evolution of the World Wide Web, have increased the need for tools that effectively support users in retrieving, understanding and mining information and knowledge contained in these data.

Formal Concept Analysis play a very important role and help fill this gap thanks to its simplicity, proficiency and versatility.

Concept lattice is used in conceptual data processing. Concept lattices is a principled way of representing and visualizing the structure of symbolic data that emerged from Rudolf Willes efforts to restructure lattice and order theory in the 1980s.

Formal Concept Analysis has become a standard technique in data and knowledge processing that has given rise to applications in data visualization, data mining, information retrieval and knowledge management.

Formal Concept Analysis is different from statistical data analysis in that the emphasis is on preference structural similarities, such as extract relation from the data description, and not on mathematical manipulations of probability distributions. By using Formal Concept Analysis one can turn any collection of objects described by a set of attributes into a lattice of concepts, where each concept covers a subset of the objects contained in the given collection and is described by the attributes shared by the objects relating to that concept. Such a conceptual representation, termed Concept Lattice, can be extracted from different types of data (e.g., text, structured data) and can then be used to support different kind of content management task.

The general impression is that Formal Concept Analysis is a powerful technology for content processing, the potentials of which have not been fully exploited to date. There are many interesting applications that could benefit from the kind of content processing performed by Formal Concept Analysis.

A lot of studies suggest that graphical representation and display of searched results can improve information retrieval performance [15, 16, 17, 18]. A graphical information display can provide a broad and concise representation of the searched results from which can quickly comprehend their relevance and importance. The graphical information display can improve the low precision and low recall of the searched results. Other similar findings have been found in a recent study [15], in which a detailed survey is presented on how visualizations can enhance information retrieval by allowing searchers to browse through a graphical representation of the requested documents. It is therefore of utmost importance for an information retrieval system to equip with a good graphical user interface that organizes the

information into an effective visual structure for the searchers to browse through during the information retrieval process.

Concept lattices have proved useful in many areas, such as knowledge representation [1], information retrieval [2], web document management [3,4], software engineering [5] and bioinformatics [6,11]. The structure of the lattice is of particular importance in these applications, i.e. the Hasse diagram of the concept lattices. For example, the immediate predecessors and successors of a concept are used in browsing web documents [4] or to infer the class hierarchy of a program [5]. The edge information of Hasse diagrams can be used to compare two concept lattices in which gene expression information has been coded [6].

A lot of researchers have proposed the use of formal concept analysis and concept lattice as a way of how to organize, structure and visualize data in information retrieval systems[7,8,9].

Using concept lattice for visualizing has its disadvantage called, dimension. The computed lattices very often do not fit in the screen even for small input data. To visualize large structures, researchers have developed interfaces that allow multiple local and global views [22]. However, the method has the disadvantage that the searchers need to map different graphical representation extended the work to adopt a variant of the Fisheye view technique to show individual nodes of the lattice on a standalone symbolic lisp machine. Despite these many research efforts, the display and comprehension of the lattice associated with a large database remains an open problem. It was the reason why researchers have developed different graphical representations [10, 11].

1.1 Goal of research

Using concept lattice for visualizing has negative points, the dimension of the computed lattices very often do not fit in the screen even for small input data. Many research have developed interfaces that allow multiple local and global views, but this method consumes time and capacity.

In this work I deal with the possible usage of matrix decompositions to improve concept lattice reduction.

The goals is:

- Minimize the input data before constructing the concept lattice.

The reduced input data has lower dimension than the original data and computing the lattice leads to creating smaller graphic representation in less time.

- Use the matrix decompositions to improve concept lattice reduction.

Use two methods of matrix decomposition, Singular Value Decomposition (SVD) and Non-Negative Matrix Factorization (NMF).

- Show difference behavior of the concept lattice reduction.
Present the different behavior of concept lattice reduction before and after using methods of matrix decompositions.

- Using matrix decomposition and concept lattice in search result clustering.
Presents and describe uses of matrix decomposition and concept lattice as a mathematical method to reduce a big value of objects in search result clustering by combining the attributes of these objects.

1.2 History of Formal Concept Analysis

1.2.1 The relationship of FCA notions to similar notions in other fields

Formal Concept Analysis defined the Formal Concepts as tool that describe a natural feature of information representation, which considered as basic elements to hierarchal and object/attributes structures as set theory or relational algebra are for relational databases. Earlier 1968 Gerard Saltons [70] mentioned that the essential concepts of lattice in the sense of FCA are documents/terms lattices.

Galois Lattices which considered as equivalent to concept lattices described by Barbut Monjardet (1970) [71].

In 2002 again independently discovered essentially concept lattices by Shreider and (cf. Gusakova Kuznetsov)[72].

Feature structure lattices as used in linguistic componential analysis are very similar to concept lattices (cf. Dyviks (1998) work)[73]. Last but not least, Barwise Seligmans (1997)[74] classifications are also concept lattices.

Wille (1982) had developed concept lattice theory with many application show the different between the independent discoveries and Formal Concept Analysis.

Wittgenstein, Rosch (1973) [75] and others says that Formal concepts in FCA can be seen as a mathematical formalization of what has been called the classical theory of concepts in psychology/philosophy, which states that a concept is formally definable via its features.

Medin (1989, p. 1476) [76] states: despite the overwhelming evidence against the classical view, there is something about it that is intuitively compelling. Even though from a psychological viewpoint the classical view does not accurately represent human cognition, the classical theory nevertheless dominates the design of computerized information systems because it is much easier to implement and to manage in an electronic environment. The classical view implicitly underlies many knowledge representation formalisms used in AI and in traditional information retrieval and library systems. Even if non-classical approaches are implemented (such as cluster analysis or neural networks), the resulting concepts are still sometimes represented in the classical manner.

Formal Concepts is a concept in Formal Concept Analysis that used to avoid the confusion with non-classical theories or non-mathematical versions of the classical theory, which is a mathematical entity. Formal concept Analysis is used in a formal domain (such as in software engineering). But if FCA is to be used in domains that are primarily concerned with human cognition, such as psychology or linguistics, the same amount of careful modeling and caution is required for FCA as is required for statistical methods in these domains.

According to FCA, Freges notions might be better translated as denotation for Bedeutung and connotation for Sinn (cf. Priss (1998) [77] for a more detailed discussion). Philosophers might disagree with these definitions but, in general, mathematical formalizations as achieved by FCA always only approximate non-formal notions held in non-mathematical disciplines. The advantage of formalizations, however, is that notions are defined with absolute precision within the formal realm and that they therefore may be implementable in software.

1.2.2 FCA in Information Retrieva

In (1968) Salton [75] tries to use lattice for information retrieval with other respect to document/ term lattices and lattices of Boolean query combinations (cf. Priss (2000) [78] for a summary of these early attempts). But he did succeed.

Godin et al. (1989) [79] has developed an information retrieval system based on document/term lattices, which was based on text without graphical representation of the lattice, he discusses Fisheye and other techniques suitable for visualizations. His group compared information retrieval based on concept lattices to Boolean queries and to navigation in hierarchical classifications (Godin et al. 1993a) [80] and concluded that the performance between Boolean queries and lattice navigation was similar and both better than the use of hierarchical classification.

Some of the researchers shift from retrieval of software components rather than information retrieval.

Mili et al. (1997) [81] discovered that the use of faceted classification is not advisable for software component retrieval because the cost of developing such classifications outweighs any benefits. Furthermore, according to them, controlled vocabularies (as used in a faceted classification) may be too restrictive with respect to programming languages. While these results are interesting, they should probably be considered with caution because, at least to our knowledge, no one outside of Godins group ever tried to replicate any of these experiments.

Carpineto Romanos (1993)[82] research was initially influenced by Godins work but has since then been independently advanced to a high level. Their Credo engine (Carpineto Romano, 2004b) [83] facilitates a lattice-based meta-search of Google results. An overview of their work and FCA applications in information retrieval in general can be found in Carpineto Romano (2004b) [83].

Carpineto Romano argue that FCA can serve three purposes in information retrieval: first, FCA can support query refinement. Because a document/term

lattice structures a search space into clusters of related documents, lattices can be used to make suggestions for query enlargement in cases where too few documents are retrieved and for query refinement in cases where too many documents are retrieved. Second, lattices can support an integration of querying and navigation (or browsing). An initial query identifies a start node in a document/term lattice. Users can then navigate to related nodes. Further queries are then used to prune a document/term lattice to help users focus their search (Carpineto Romano, 1996a) [84]. Third, a thesaurus hierarchy can be integrated with a concept lattice - an idea which was independently discussed by different researchers (Carpineto Romano (1996b) [85], Skorsky (1997)[86] , Priss (1997))[87] but is probably still not sufficiently resolved.

Apart from Credo, a second FCA application that has reached professional quality is the Mail-Sleuth software (Eklund et al., 2004) [88]. The development of this software is based on earlier research on retrieval of information from semi-structured texts (Cole Eklund (2001) [89] and Cole Stumme (2000))[90].

In general, FCA software appears to show a promise for applications in information retrieval, however, with a few limitations. Similar to latent semantic analysis, LSA (Dumais, 2004) [91], FCA is not suited for direct manipulation of very large data sources. It is difficult to give precise upper limits because it depends on the application. It also matters whether both the object and attribute sets are large or only one of them. FCA has been applied to thousands of documents in a small library (Rock Wille, 2000) [92]. But presumably, it would not be possible to apply FCA (or LSA for that matter) directly to the complete Google database. But either method can be applied as a secondary tool to reorganize a set of documents resulting from a Google query (as demonstrated by the Credo engine). Since both FCA and LSA employ matrices, although LSA matrices are many-valued contexts in FCA terminology, it would be interesting to compare both methods more closely, which, at least to our knowledge, has not yet been done. FCA technology claims to be human-centered due to its philosophical basis but only few practical usability studies exist (such as Eklund et al. (2004)) [88]. It is to be hoped that at least Credo will be extensively tested for usability.

1.2.3 FCA as a tool for knowledge representation and knowledge discovery

Formal Concept Analysis provides a contrast to some of the traditional, statistical means of data analysis and knowledge representation because of its focus on human-centered approaches. Wille (1982)[93] explains that he was influenced by H. von Hentigs (1972) [94] concerns about the status of sciences in the modern world. Hentigs idea was to restructure theoretical developments in order to integrate, rationalize origins, connections, interpretations, and applications (Wille, 1982, p. 447) [93]. FCA was started as an attempt at restructuring mathematical lattice theory in a manner that both facilitates communication about mathematical theory to a wider non-mathematical audience and facilitates exploitation of

mathematical theory for a wide range of applications. The concept lattices of FCA serve as a means for communication, exploration and discussion which compiles both with Habermass Theory of Communicative Action and Peirces pragmatism (cf. Wille, 1997a) [95].

They facilitate discussion and exploration of conceptual structures, concept lattices can be characterized as a means of external cognition in the sense of Scaife Rogers (1996) [96]. The use of diagrams for reasoning has been formally investigated by Dau (2004) [97]. He observes that mathematicians often include diagrams in their descriptions of mathematical facts but that normally such diagrams are not permissible as arguments themselves. By formally distinguishing between a mathematical structure and its diagrammatic representation, Dau provides a framework in which diagrams can be used for formal reasoning. Thus in addition to an intuitive notion of the importance of visualizations, such as concept lattices, Dau can even formally evaluate their usefulness within a formal framework itself.

FCA has been examined with respect to principles of knowledge representation. Wille (1997a) [95] identifies ten functions of knowledge processing (exploring, searching, recognizing, identifying, analyzing, investigating, deciding, improving, restructuring and memorizing) and investigates how these are supported by FCA. Stumme (2002) [98] analyses FCA with respect to Davis et al.s (1993) [99] five principles of knowledge representation: knowledge representations as a medium of human expression, a set of ontological commitments, a surrogate, a fragmentary theory of intelligent reasoning, and a medium for pragmatically efficient computation.

Conceptual Knowledge Discovery (Hereth et al. (2000) [100] and Stumme et al. (1998)) [101] is mainly supported by Toscana systems (Kollewe et al., 1994) [102]. In contrast to statistical software, which attempts to provide probable answers to narrow questions, Toscana systems facilitate browsing and interactive exploration of implicit and explicit structures. Because the preparation of data for input into a Toscana system is labor-intensive and requires substantial knowledge of FCA, Toscana systems are usually compiled by an FCA expert in co-operation with a domain expert. Wille (2001) [103] argues that this is an advantage because the processes involved in creating a conceptual representation (in the sense of FCA) encourages the discovery of implicit information and facilitates the conversion of information into knowledge. Nevertheless, the effort required for setting up Toscana systems may be a reason why their use is not more wide-spread. It should be emphasized, however, that only the preparation of a Toscana system requires expertise. End-users can utilize such a system after reading a brief introduction. A side effect of the careful set-up of a Toscana system is that it can be less error prone than some statistical methods because a careful conceptual modeling prevents data misrepresentation.

Other researchers focus on algorithmic issues (such as Kuznetsov Obiedkov (2002)) [104] or abstract issues (such as Wille (2001)) [103]. These issues are important but it would be more interesting to see more realistic applications. Furthermore, it might be interesting to compare relational scaling as described by Prediger Stumme (1999) [105] to methods employed in business intelligence and

data warehousing because they appear to pursue similar goals.

1.2.4 Applications of FCA in logic and AI

Since about 1996, attempts have been made to combine FCA with other formalisms of conceptual structures, such as Sowas (1984) [106] Conceptual Graphs.

Wille (1997b) [107] describes a translation of Conceptual Graphs into formal contexts and concept lattices. Mineau et al. (1999) [108] investigate the commonalities between both theories at a more general level. Conceptual Graphs (Sowa, 1984) [106] are formalism for knowledge representation, which is similar to semantic networks, entity relationship diagrams and the Semantic Web standard RDF. Sowa developed his Conceptual Graphs based on Peirces Existential Graphs, a graphical, symbolic notation for reasoning which incorporates aspects of context and modalities. In contrast to the hierarchical relations which are expressed in concept lattices, Conceptual Graphs can be used to represent semantic relations, such as part/whole, and linguistic case relations, such as Agent (or subject of a sentence), Patient (or direct object of a sentence), and so on. Natural language sentences can be translated more or less directly into conceptual graphs. A connection with FCA is established via types: each concept of a Conceptual Graph contains information about its type. The types form a hierarchy which can be modeled with FCA. According to Wille (1997b) [107] a combination of FCA and Conceptual Graphs can facilitate a formalization of Elementary Logic and thus presents a powerful formalism for the representation and analysis of human reasoning and argumentation.

Apart from conceptual graphs, connections have been established between FCA and Description Logics (Prediger Stumme, 1999) [105]. In contrast to Conceptual Graphs and FCA, which primarily focus on representations, Description Logics investigate expressivity and computability of logical representations.

A combination of FCA and Conceptual Graphs is not just ontology formalism because of its philosophical foundation. Wille (2000a) [109] perceives logic (and human reasoning and argumentation) as a Kantian triad of concepts, judgments and conclusions. The goal is to use FCA to achieve a mathematician of these three philosophical doctrines in a framework of contextual logic (Prediger, 1998) [110]. While FCA is used to mathematize concepts, Conceptual Graphs are used to mathematize judgments. A combination of Conceptual Graphs, FCA and Description Logics can then be used to mathematize conclusions. Wille sees 15this as a continuation of Booles logic of signs and classes (Wille (2000b) and (2004)) [111],[112]. But in contrast to Boole, who envisioned a Universal set or class, FCA focuses on formal contexts, which are finite in most applications (Ganter Wille, 1999b) [113] and which avoid some of the confusion caused by the assumption of universality.

A challenge for this kind of mathematization of logic is the treatment of negation (cf. Wille (2000b) [111], Dau (2000) [114], Kwuida et al. (2004)) [115].

With respect to concept lattices it can occur that negations of formal concepts

are not materialized as formal concepts themselves. This is due to the fact that a formal context does not normally explicitly specify negation. If an object does not have an attribute assigned, it can mean that the attribute is irrelevant, the relationship is unknown or that the object does sometimes, usually or never have the attribute (Burmeister Holzer, 2000) [116]. Therefore, adding negation to a concept lattice requires considering so-called proto-concepts or semi-concepts (Wille 2000b) [111], which are mathematical structures similar to formal concepts but whose mathematical properties are more complex and more difficult to describe. The study of the structures arising from such proto- or semi-concepts is on-going (eg. Hereth Correia Klinger, 2004)[117]. Other aspects of contextual logic pertain to implicit knowledge, incorporation of background knowledge (Hereth Correia Klinger, 2004) and the incorporation of existential quantifiers (Wille, 2002)[118].

1.3 Thesis outline

CHAPTER TWO
Introduces the basic terms of Formal Concept Analysis (FCA), and serves as an introduction to the theory of concept lattices. It formalize the notions of concept, context and concept lattice.

CHAPTER THREE
Introduction to matrix decomposition and review how to use matrix decomposition for data analysis and a large set of different decompositions.

CHAPTER FOUR
Describes how singular value decomposition used to transforms the data matrix in a way that exposes the amount of variation in the data relative to a set of latent features. Also present SVD and Principal Component Analysis.

CHAPTER FIVE
Describes non-negative matrix factorization and how we can use it for decomposition of matrix.

CHAPTER SIX
Presents how I apply matrix decomposition in concept lattice, how it improved factor to Concept lattice reduction. There are some examples to illustrate my approach. This show how we can use FCA on large data collections and visualize data structure via concept lattice.

CHAPTER SEVEN
Presents and describe my uses of matrix decomposition and concept lattice as a mathematical method to reduce a big value of objects in search result clustering by combining the attributes of these objects.

CHAPTER EIGHT
Contains a summary and conclusion of this work.

CHAPTER NINE
Present a publications of the author.

CHAPTER TEN
Present a references which used in this work.

2 Concept Lattices

Formal Concept Analysis (FCA) is a general data analysis method based on the lattice theory. FCA was introduced in 1982 by Wille [93]. The basic algorithms for concept lattice computation where published by Ganter in 1984 [121]. Carpineto and Romano summarized in [3] both the mathematical and computer scientific (with focus on information retrieval) perspective of the FCA. Good overview of the recent state was written also by Priss [122]. There has been several approaches for the fuzzification of FCA. Belohlavek And Vychodil have reviewed and compared these approaches in [123].

Since the beginning, the concept lattice can be visualized using so-called Hasse diagram (originating back to Vogtin 1895). The automated drawing algorithms are introduced in [30]. As it is a non-trivial procedure, many heuristics addressing different a esthetic criteria can be applied [31]. Eklundet.al., using a particular application, in [124] shows that such type of diagram can be (under Some conditions) understood also by inexperienced users.

Applications of FCA covers a wide range of different fields, such as linguistics[125], source code analysis [126], mail box navigation [8], image analysis [127], blogosphere analysis [128] and recommendation system [129].

This part serves as an introduction to the theory of concept lattices. It formalize the notions of concept, context and concept lattice. This text is quoted from book Concept Data Analysis: Theory and Applications[3].

2.1 Basic notion of orders and lattices

A binary relation $\leq$ on set P is called Partial order relation on P if, for all $x, y, z \in P$:

1. $x \leq x$,
2. $x \leq y$ and $y \leq x$ imply $x = y$,
3. $x \leq y$ and $y \leq z$ imply $x \leq z$.

A set P equipped with an order relation $\leq$, denoted by $(P, \leq)$ or just by P, is called an ordered set (or partially ordered set or simply order).

Examples of ordered sets are the power-set $\rho(X)$ of all subsets of any set X with set inclusion and the real numbers $\Re$ with the usual $\leq$ relation. The latter

ordered set is a chain, because for all $x, y \in \Re$, either $x \leq y$ or $y \leq x$ (i.e., any two element of $\Re$ are comparable).

Let P be an ordered set and let $x, y \in P$. Element x is covered by y (or y covered x) if

$$x < y \text{ and there is no } z \in P \text{ such that } x < z < y.$$

We write $x \prec y$. We may also say that x is a lower neighbor of y, or that y is an upper neighbor of x. If P is finite, $x < y$ if and only if there exists a finite sequence of covering relation between x and y.

Every finite ordered set $(P, \leq)$ can be drawn. We use small circles to represent the elements of P and interconnecting lines to indicate the covering relation. The drawing is such that element x is placed below element y if $x \prec y$. This representation is called line diagram or Hasse diagram.

Cleary, the same ordered set may have many different diagrams, and the notion of a good line diagram is difficult to formalize. Figure 2.1 show several possible ways to draw the same ordered set.

From a line diagram we can easily tell whether one element is less then another: $x < y$ if and only if there is an ascending path from x to y. Figure 2.2 presents line diagram for all ordered sets with three elements.[19]

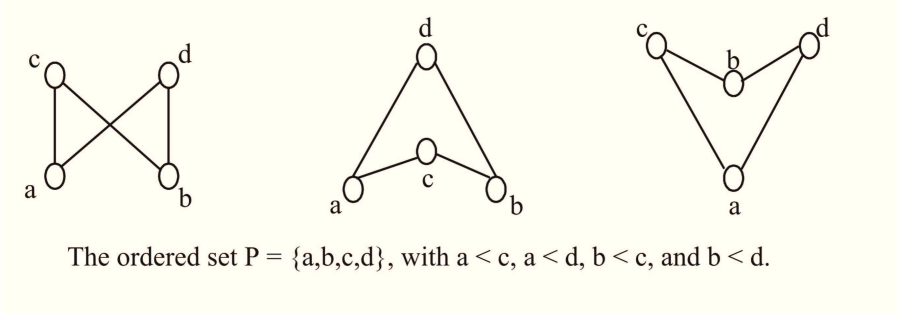


Figure 2.1: Alternative line diagram

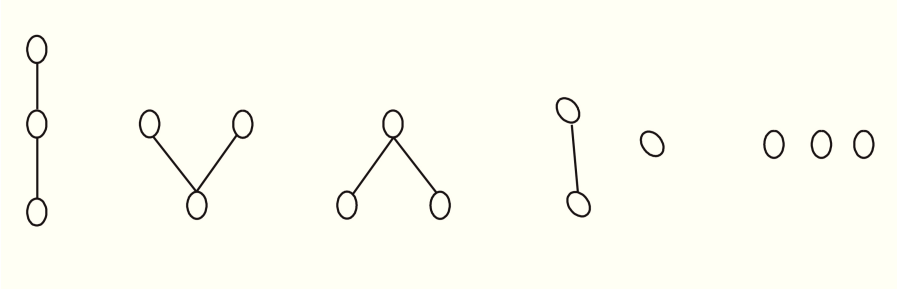


Figure 2.2: Line diagram of all possible ordered sets with three elements

The greatest element of P, is called the top element of $P(\top)$; the least element of P, is called the bottom element of $P(\bot)$. For instance, d and a are the top and the bottom elements of the ordered set in Figure 2.3.

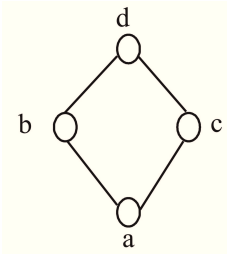


Figure 2.3: The top and the bottom elements of the ordered set

New ordered sets can be built from existing ones in several ways. Let $(P_1; \leq)$ and $(P_2; \leq)$ be disjoint ordered sets. The cardinal sum of them is the ordered set $(P_1 \bigcup P_2; \leq)$, the order relation being defined as follows: $x \leq y$ if and only if either $x, y \in P_1$ and $x \leq y$ in P_1 or $x, y \in P_2$ and $x \leq y$ in P_2.

A line diagram for the cardinal sum can be obtained by placing side by side diagram for the summands. The linear sum of $(P_1; \leq)$ and $(P_2; \leq)$ is defined as the cardinal sum except that $x \leq y$ holds if $x \in P_1$, $y \in P_2$. A diagram for the linear sum is formed by placing a diagram for P_2 directly above a diagram for P_1 and then adding a line segment from each maximal element of P_1 to each minimal element of P_2.

The direct product of two disjoint ordered sets $(P_1; \leq)$ and $(P_2; \leq)$ is the ordered set $(P_1 \times P_2; \leq)$, where $(P_1 P_2)$ is the Cartesian product of P_1 and P_2 and the order relation on the product is as follows:

$$(x_1, x_2) \leq (y_1, y_2) \leftrightarrow x_1 \leq y_1 \text{ and } x_2 \leq y_2$$

An example of a product of two ordered sets is shown in Figure 2.4.

These composition operators can be useful not only to join two or more ordered sets, but also to analyze a given ordered set as if it consisted of simpler part.

Introducing the notions of lattice and a complete lattice, which are two important classes of ordered sets characterized by the existence of certain upper bounds or lower bounds of their elements.

Let $(P; \leq)$ be an ordered set and S a subset of P. An element $x \in P$ is called an upper bound of S if $s \leq x$ for all $s \in S$. A lower bound is defined dually. The set of all upper bounds of S is denoted by S^u and the set of all lower bounds by S^l; S^u is always an up-set and S^l a down-set [20].

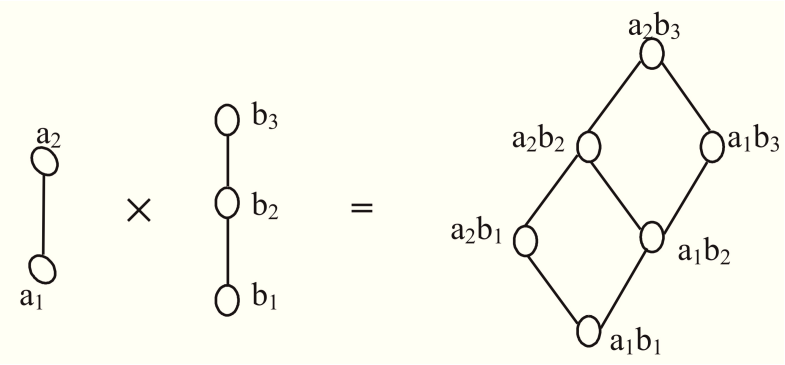


Figure 2.4: Example of a product of two ordered sets

If there is a least element S^u, it is called the least upper bound or supremum of S, denoted by sup S. Dually, a greatest element in S^l, is called the greatest lower bound or infimum of S, denoted by inf S.

Supremum and infimum are frequently also called join and meet. It write $x \vee y$ ('x join y') instead of sup x, y and $x \wedge y$ ('x meet y') instead of inf x, y. It also write $\vee S$ (the 'join of S') and $\wedge S$ (the 'meet of S') instead of sup S and inf S.

In an ordered set $P, x \vee y$ may fail to exist either because x and y have no upper bound or because they have no least upper bound. In figure 2.1, $c \vee d$ does not exist because there is no common upper bound, while $a \vee b$ does not exist as the set of upper bounds (c, d) does not have a least element.

An ordered set $(P; \leq)$ is called a lattice if for any pair of elements x and y in P the supremum $x \vee y$ and the infimum $x \wedge y$ always exist. An ordered set $(P; \leq)$ is called a complete lattice if the supremum $\vee S$ and the infimum $\wedge S$ exist for any subset S of P. Every complete lattice has a top element, also called the unit element of the lattice, and a bottom element, called the zero element [21].

A complete lattice can be characterized in other ways. Two useful propositions are that every non-empty finite lattice is complete and that a non-empty ordered set in which the infimum exists for every subset is a complete lattice.

A complete lattice L is called distributive if the following distributive laws hold for all $x, y, z \in L$:

$$x \wedge (y \vee z) = (x \wedge y) \vee (y \wedge z)$$
$$x \vee (y \wedge z) = (x \vee y) \wedge (y \vee z)$$

2.2 Context, concept, and concept lattice

A context is a triple (G, M, I) consisting of two sets G and M and a relation I between G and M. The elements of G are called the objects and the elements of M are called the attributes. The relation I is also called the incidence relation of

the context. It's written as gIm or $(g, m) \in I$ to mean that the object g has the attribute m. We may think of the set of attributes associated with an object as a bit vector; each bit corresponds to a possible attribute and is on or off depending upon whether an object has that attribute [22,23].

In table 2.1 shown a context for some objects and the attributes are properties describing the objects. A cross in the ij position of the table indicates that object i is described by attribute j, or, equivalently, that attribute j describes object i.

	A1	A2	A3	A4	A5	A6	A7	A8	A9
O1		x			x	x		x	
O2		x	x		x	x			
O3				x	x			x	
O4	x		x		x		x		
O5			x		x	x	x		
O6	x				x		x		
O7	x				x		x		x

Figure 2.5: A context for objects

For a set $A \subseteq G$ of objects define:

$$A' = (m \in M | gIm \text{ for all } g \in A)$$

Correspondingly, for a set $B \subseteq M$ of attributes define:

$$B' = (g \in G | gIm \text{ for all } m \in B)$$

In other words, A' is the set of attributes common to the objects in A, while B' is the set of objects which have all attributes in B.

A concept of the context (G, M, I) is a pair (A, B) where

$$A \subseteq G,\ B \subseteq M,\ A' = B, \text{ and } B' = A.$$

A is the extent and B the intent of the concept (A, B). The set of all concepts of the context (G, M, I) is denoted by $C(G, M, I)$. Thus, a concept is considered to be identified by its extent and its intent: the extent consists of all objects belonging to the concept while the intent contains all attributes shared by the objects. Essentially, it refers to the inverse relation between the number of attributes that are necessary to describe a concept and the number of objects to which the concept applies [24].

The concepts of a context also permit a geometrical interpretation, because they can be seen as maximal rectangles of the table representing the context. More precisely, a maximal rectangle of the context (G, M, I) is a pair (A, B) where $A \subseteq G$, $B \subseteq M$, and such that

$$\forall g \in G \setminus A, \exists m \in M | (g, m) \notin I$$

and

$$\forall m \in M \setminus B, \exists g \in G || (g, m) \notin I$$

The last two conditions require that for each object not included in the maximal rectangle there exists at least one attribute of the maximal rectangle that is not shared by the object, and, correspondingly, that for each attribute not included in the maximal rectangle there exists at least one object of the maximal rectangle that does not share the attribute.

An order relation on the set of concepts of a context is defined in the following way. If (A_1, B_1) and (A_2, B_2) are concepts in $C(G, M, I)$, we say that (A_1, B_1) is a subconcept of (A_2, B_2), or that (A_2, B_2) is a superconcept of (A_1, B_1), and we write $(A_1, B_1) \leq (A_2, B_2)$, if $A_1 \subseteq A_2$ (which is equivalent to $B_1 \supseteq B_2$). We can also say that (A_1, B_1) is smaller (or more specific) than (A_2, B_2), or that (A_2, B_2) is larger (or more general) than (A_1, B_1). The relation $\leq$ is called the hierarchical order (or simply order) of the concepts [25].

The ordered set $C(G, M, I; \leq)$ is called the concept lattice (or Galois lattice) of the context (G, M, I). It can be characterized by the following theorem.

The Basic Theorem on Concept Lattices. Let (G, M, I) be a context. Then $C(G, M, I; \leq)$ is a complete lattice in which join and meet are given by:

$$\vee_{j \in J}(A_j, B_j) = ((\bigcup_{j \in J} A_j)^{"}, \bigcap_{j \in J} B_j),$$

$$\wedge_{j \in J}(A_j, B_j) = (\bigcap_{j \in J} A_j, (\bigcup_{j \in J} B_j)^{"}).$$

Turning to non-mathematical examples, in figure 2.5 shows the concept lattice of the objects context given in table 2.1. The concept lattice contains as many as 16 concepts, including a top element with a non-empty intent (because all the objects have the attributes number 4) and a bottom element with an empty extent (because there is no object sharing all attributes).

In fact, as a concept lattice contains only the pairs that are complete with respect to I according to the given definition, there are several subsets of properties that cannot be admissible intents or extents for the given context.

The similarities and the diversities of the objects described in the context are well represented in the concept lattice. On closer inspection, some concepts in figure 2.5 correspond to well-known objects groups such as concept 3 (a group

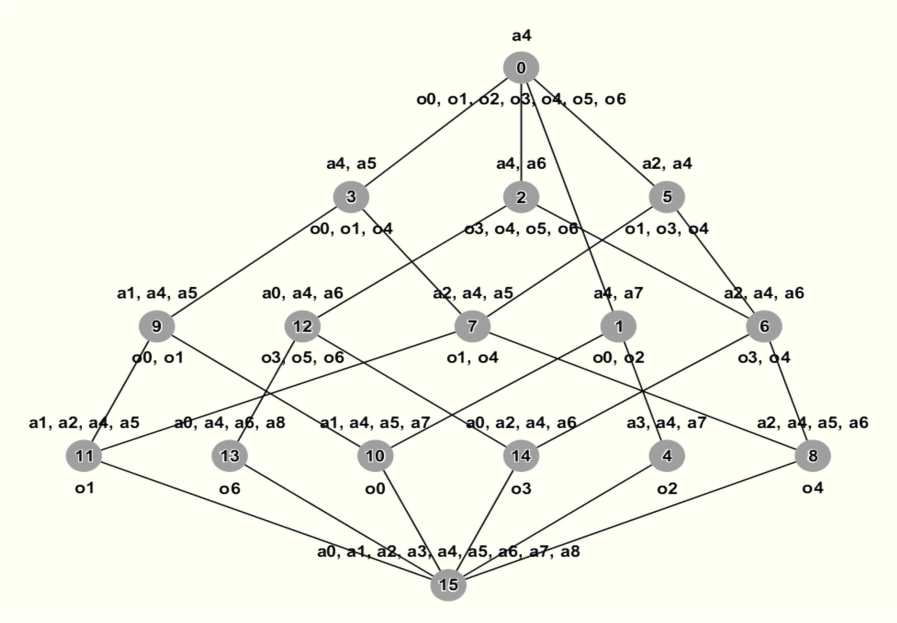


Figure 2.6: Concept lattice for the context of table

which attributes A4 and A5), concept 12 (a group which attributes A0, A4 and A6).

In a concept lattice, it is as if many possible hierarchical classifications (composed of overlapping data sets) existed in parallel with each other with no duplication of information (as opposed to decision trees, for instance, where the same subtree may occur multiple times over the whole structure). The concept lattice thus represents a formalism for exploring such hierarchies for correlations, similarities, refinements, anomalies, or even inconsistencies.

2.3 Algorithms

This part presents the question of the efficient construction of concept lattices. After analyzing the size of the lattice as a function of the input size, we will present a number of algorithms catering for the most common situations that can be encountered. More specifically, there are three main tasks: (i) determination of the whole set of concepts; (ii) batch construction of the full concept lattice including the Hasse diagram; and (iii) generation of partial concept lattices for tasks where it is sufficient to generate a limited portion of the whole structure [26].

2.3.1 Computational space complexity of concept lattices

There are some theoretical upper bounds on the number of concepts present in the lattice that depend on the size of the input context. If a context is described

by $|G|$ objects and $|M|$ attributes, the corresponding lattice will contain at most $2^{|G|}$ or $2^{|M|}$ concepts, whichever is the largest.

It turns out that such high theoretical bounds can be reached for some actual contexts. A well-known example is the $(n \times n)$-dimensional context containing ones in all positions, but zeros along the diagonal; the corresponding lattice contains exactly 2^n concepts [27].

This situation may occur on rare occasions. It is also often the case that the number of attributes per object has upper bound given by a constant K. In this case, since each new object can generate at most 2^K concepts (i.e., all possible subsets of its intent), the number of concepts is bounded by $|G|.2^K$. With this assumption, the growth of the number of concepts is therefore linear with the number of objects, but the factor may become very large even for relatively small values of K.

2.3.2 Construction of the set of concepts

This part describe three basic algorithms for finding the set of concepts associated with a given context. Although they may not be the best available algorithms from an application point of view, they very clearly illustrate some fundamental computational problems presented by the task at hand. In addition, they may be used as the building blocks for more powerful algorithms[28].

The Naive algorithm

A subset A of G is the extent of some concept if and only if $A'' = A$, in which case the unique concept of which A is an extent is (A, A'). Dually, a subset B of M is the intent of some concept if and only if $B'' = B$, the unique concept of which B is an intent being (B', B). Thus, the simplest algorithm for generating the set of concepts of a formal context (G, M, I) would be to form (A'', A') for all $A \subseteq G$, or (B', B'') for all $B \subseteq M$. This method is extremely inefficient as it requires the consideration of all the subsets of G or M together with their corresponding closures, regardless of the size of the concept lattice being generated.

Ordering subsets based on their closure

A better strategy for finding the set of concepts by using closures is based on ordering the subsets of G in such a way that only some subsets need to be examined.

After imposing a linear order, denoted by, $(< G)$, on the elements of G, all subsets of G may, in turn, be strictly ordered according to a particular lexicographic

order ($\preceq$) defined as follows. A subset A of G is smaller (according to $\preceq$) than a subset B of G if the smallest element which distinguishes A and B according to ($< G$) belongs to B. Note that this is an order on bit strings, not on natural numbers.

From a computational point of view, the lexicographic order can be generated in the following way. The next subset is generated from the current subset in two steps, i.e., by adding the maximum object of G that is not contained in the current subset and then deleting all objects in the current subset that are greater than the object just added.

This particular enumeration of subsets of G is well formed with respect to the properties of closures. It turns out that, for any subset of objects A, if $A'' \setminus A$ contains only elements that are greater ($< G$) than those in A, then all subsets that follow A and precede A'' in the lexicographic order $\preceq$ will produce a concept extent that is either equal to A'' or will be generated later by some subset following A''. Thus, in this case it is possible to skip examination of some of the subsets of G. If the condition about the difference between A'' and A is not satisfied, then the creation of the concept with extent A'' will be postponed until the subset A'' is encountered, thus avoiding multiple generation of the same concept.

The algorithm considers one current subset A in the lexicographic order at a time, until the last element of the order (i.e., the subset G) is generated. If A'' does not contain objects that are smaller than those in A, which is trivially true if $A'' = A$, then the concept with extent A'' is added to the set of concepts and the next subset of G to be examined is set to the successor of A''. If the condition is not satisfied, the concept with extent A'' is not added to the list of concepts because it will at the latest be generated when examining the subset A'', and the next subset in the lexicographic order is generated.

The following algorithm illustrate the *Next closure* algorithm, as well as the other algorithms which will be presented below, we will be using the context in table 2.1[3].

Next Closure
Input: Context(G, M, I)
Output: The set C of all concepts of (G, M, I)

1. $C := (M', M)$
2. $currSubset := max(g \in G)$
3. $nextObj := max(g \in G)$
4. **while** $currSubset \neq G$
5. **if** there is no $g \in currSubset'' \setminus currSubset$ such that $g < nextObj$
6. **then**
7. $C := C \bigcup (currSubset'', currSubset')$
8. $nextObj := max(g \in G \setminus currSubset'')$
9. $currSubset := currSubset''$
10. **else**
11. $nextObj := max(g \in G \setminus currSubset$ such that $g < max(currSubset))$
12. $currSubset := currSubset \bigcup (nextObj)$
13. $currSubset := currSubset \setminus (g \in currSubset$ such that $nextObj < g)$

The number of closures generated will therefore usually be greater than the number of concepts and smaller than the number of subsets, although in the worst case the algorithm may have to explore all subsets of G. An example is the context where each object has the same description; the concept lattice consists of the single element (G, M) and the *Next Closure* algorithm must generate all the subsets preceding G. The worst case time complexity of the algorithm is thus $O(2^{|G|}|G||M|)$, where $O(|G||M|)$ is the time required to compute the prime ($'$) and double prime ($''$) operators.

One advantage of this algorithm is that every concept is created only once, and thus no extra space and time is needed to store and retrieve the generated concepts. A main limitation, besides its relative efficiency, is that the algorithm does not support the construction of the Hasse diagram, for there is no relationship between the lexicographic order and the order defined over the concepts in the lattice.

Computing intersections incrementally

A different solution to the problem of finding all the concepts is based on the observation that every concept extent is the intersection of attribute extents and every concept intent is the intersection of object intents.

To generate the set of concepts, it is therefore sufficient to form all possible intersections between the sets of objects associated with each attribute and then use the context to find the intent corresponding to each generated concept extent. Alternatively, we could calculate all intersections of object intents and then determine the extent of each such concept intent.

In order to form all possible intersections, it is convenient to consider one attribute (object) at a time and calculate the intersection between its extent (intent) and each concept extent (intent) in the current set of concepts, where the current set of concepts contains concepts generated by the attributes (objects) that have already been examined.

The following algorithm is a pseudo-code for the determination of all concepts. This version iterates over the set of objects; it is termed Object Intersections[3].

Object Intersections
Input: Context(G, M, I)
Output: The set C of all concepts of (G, M, I)

1. $C := (M', M)$;
2. **for** each $g \in G$
3. **for** each $(X, Y) \in C$
4. $Inters := Y \bigcap g'$;
5. **if** $Inters$ is different from any concept intent in C **then**
6. $C := C \bigcup (Inters', Inters)$

The major drawbacks of the Object Intersections algorithm are that the same concepts can be generated several times and no provision is made for generating the concept lattice. The worst-case time complexity is as follows. Line 4 can be done in $O(|M|)$. Line 5 would take $O(|M||C|)$ time, but its cost can be significantly reduced by maintaining a search tree or other data structure for set manipulation problems. One very useful search structure, is a trie. A trie, pronounced 'try', is essentially an N-ary tree whose nodes are N-place vectors with components corresponding to digits or characters. Each node on level l represents the set of all keys that begin with a certain sequence of l characters and specifies an N-way branch, depending on the $(l + 1)$th character [29].

2.3.3 Construction of concept lattices

The algorithms introduced in the previous section are able to determine the set of concepts of a given context but they are unable to find the line diagram at the same time. This section address the problem of building the full concept lattice, which is a prerequisite for many applications.

Finding neighbors

One of the best-known and conceptually simplest algorithms to construct the set of concepts along with the Hasse diagram is based on generating neighbors iteratively, according to the $\prec$ *relation*.

Starting from the top element of the lattice (G, G'), the algorithm builds one level at a time, where the next level contains the children of all concepts present in the current level. More specifically, for each concept in the current level, a function (described below) is invoked that calculates the lower neighbors of that concept; then it is checked if each returned child has not been already generated, in which case the concept is added to the lattice, and the concept is finally linked to its parent.

The sequence of concepts generated this way corresponds to a top-down breadth-first visit of the final lattice, although a depth-first implementation would work equally well. The following algorithm is a pseudo-code of *Next Neighbors*. For the sake of generality, the concept lattice is seen as a set of concepts C and of a set of edges E, where the edges are ordered pairs of concepts (c_1, c_2) such that $c_1 \prec c_2$, i.e., c_1 is a lower neighbor of c_2. In practice, however, it is convenient to implement each concept as a record with pointers to its neighbors[3].

NextNeighbours
Input: Context (G, M, I)
Output: The concept lattice $L = (C, E)$ of (G, M, I)

1. $C := (G, G')$
2. $E := \oslash$;
3. $currentLevel := (G, G')$
4. **while** $currentLevel \neq 0$
5. $\quad nextLevel := \oslash$
6. $\quad$ **for** each $(X, Y) \in currentLevel$
7. $\quad\quad lowerNeighbors := FindLowerNeighbours(X, Y)$
8. $\quad\quad$ **for** each $(Xl, Y1) \in lowerNeighbours$
9. $\quad\quad\quad$ **if** $(Xl, Y1) \notin C$ **then**
10. $\quad\quad\quad\quad C := C \bigcup (Xl, Y1)$
11. $\quad\quad\quad\quad nextLevel := nextLevel \bigcup (Xl, Y1)$
12. $\quad\quad\quad$ add edge $(X_l, Y_1) \rightarrow (X, Y)$ to E
13. $\quad\quad currentLevel := nextLevel$

function $FindLowerNeighbours(X, Y)$
/* Returns the lower Neighbors of a concept * /
1. $candidates := \oslash$
2. **for** each $m \in M \setminus Y$
3. $\quad X_l := (Y \bigcup (m))'$
4. $\quad Y_1 := X_1^{'}$
5. $\quad$ **if** $(X_l, Y_1) \notin candidates$ **then**
6. $\quad\quad candidates := candidates \bigcup (X_l, Y_1)$
7. **return** maximally general *candidates*

The calculation of the children of a given node is performed by the function *FindLowerNeighbors*. The implementation of this function is based on the observation that all lower neighbors of a concept (X, Y) are contained in a small set of concepts, each of which is formed by adding just one new attribute m to $Y (Y_1 = Y \bigcup (m))$ and then by computing $(Y_1^{'}, Y_1^{''})$. At this point, by comparing the candidates generated, one could select only those with the largest $Y_1^{'}$, thus obtaining the actual set of lower neighbors.

If an auxiliary search tree is used to check whether a concept has already been generated (line 9), the worst-case time complexity of the algorithm is obtained by multiplying the number of invocations of the function *FindLowerNeighbors* (which is equal to the total number of concepts) by the cost of computing the lower neighbors of a given node. The latter requires the calculation of the prime operator for generating the candidates (line 3 and 4 in function *FindLowerNeighbors*), which takes $O(|G||M|)$ time, for at most $|M|$ times. As the final removal of non-maximal candidates takes at most $O(|G||M|^2)$ time, the overall complexity of the algorithms is still $O(|C||G||M|^2)$.

Using the concepts to find the edges

In the Next Neighbors algorithm, concepts and edges are created simultaneously due to the construction strategy based on the $\prec$ relation; however, the search for the neighbors of the current concept does not take advantage of the concepts that have already been generated.

An alternative method to build the full concept lattice is to create the whole set of concepts first, and then to set the edges. Although this method may seem less efficient, it has the advantage that the generation of candidate neighbors may be faster, provided that an efficient search structure is used to retrieve the concepts that have been created in the first step[3].

Concepts Cover
Input: Context (G, M, I)
Output: The concept lattice $L = (C, E)$ of (G, M, I)

1. find C with the Object Intersections algorithm
2. $CoveringEdges(C, (G, M, I))$

function $CoveringEdges(C, (G, M, I))$
/* Sets the covering edges between the concepts in C */
1. **for** each $(X, Y) \in C$
2. set count of any concept in C to 0
3. **for** each $m \in M \setminus Y$
4. $inters := X \bigcap (m)'$
5. find $(X_1, Y_1) \in C$ such that $X_1 = inters$
6. $count(X_1, Y_1) := count(X_1, Y_1) + 1$
7. **if** $(|Y_1| - |Y|) = count(X_1, Y_1)$ **then**
8. add edge $(X_1, Y_1) \rightarrow (X, Y)$ to E

The above algorithm is the pseudo-code of the *Concepts Cover algorithm*. The top-level description consists of computing the set of concepts C, e.g., by using the Object Intersections algorithm described earlier, followed by the determination of its covering graph. The *CoveringEdges* function works as follows. One concept (X, Y) of C at a time is examined, and its lower neighbors are found. Similar to function *FindLowerNeighbors*, the candidate neighbors are generated by considering, for each attribute m not contained in Y, the concept (X_1, Y_1), where X_1 is the set of objects containing both the attributes in X and the attribute m, and Y_1 is the intent of the concept having X_1 as extent.

In this case, X_l is found by taking the intersection of X and the attribute extent of m, which is faster than the corresponding operation on line 3 of function *FindLowerNeighbors*. Furthermore, rather than computing Y_1 from X_1 via the prime operator, Y_1 is more efficiently retrieved from the concept trie using X_1 as the key.

The problem of selecting maximal candidates is also dealt with in a different way. The algorithm maintains a counter for each concept in C. At the outset each

counter is initialized to zero. The counter of a concept (X_1, Y_1) increases whenever that concept is retrieved as a candidate; as soon as the difference between the cardinalities of Y_1 and Y becomes equal to the counter of (X_1, Y_1), the concept (X_1, Y_1) is a child of (X, Y). This ensures that only maximal candidates are selected and that each maximal candidate is taken exactly once.

The worst-case time complexity of the Concepts Cover algorithm is more favorable than that of Next Neighbors. The complexity of step 1, using the Object Intersections algorithm, is $O(|G||C||M|)$. The complexity of CoveringEdges is $O(|C||M|(|G| + |M|))$, as lines 4, 5 and 7, which take $O(|G| + |M|)$ time, are executed $|C||M|$ times. The overall time complexity is thus $O(|C||M|(|G| + |M|))$.

2.3.4 Visualization

Most of the applications based on concept lattices require some form of exploration of the graph diagram on the part of the user. However, forming useful visualizations of graph structures is notoriously difficult due to the conflicting issues of size, layout and legibility on limited screen area. Furthermore, visual edge crossing may be detrimental to the comprehension of graph structures.

One main concern is thus the aesthetic of the graph layout. This objective is usually pursued by reducing visual edge crossing or by promoting visual symmetry when the graph itself has symmetrical properties; however, the final result can be evaluated only by the user. Speed is another important criterion, but it may conflict with the former because minimizing the number of edge crossings is usually a time-consuming process. Concept lattices are no exception, although we can take advantage of geometric representations of specific substructures of lattices such as cubes and diamonds which may have some value for the overall structure ([30], [31]).

Regardless of aesthetic and efficiency considerations, some graphs are just too large to fit on a screen. This is typically the case for concept lattices: except for toy databases, we cannot look at the complete set of concepts and edges on one static display. Interactive or incremental or scaling techniques are necessary.

The common approach is to let the user examine a subset of concepts and edges based on the specific task being performed and on the past interaction with the system. In the concept lattice scenario, the system should be able to show or hide parts of the lattice via interactive specification manipulation of concepts, or relationships between concepts, or even subsets of attributes.

This part present three different ways of visualizing a concept lattice that have been implemented with some variants in several system prototypes.

Hierarchical folders

Because of the difficulties associated with visualizing its graph structure, one approach is to reduce the concept lattice to a simpler structure that lends itself to more understandable visual layouts. A tree is the most obvious choice.

A concept lattice can be easily represented as a tree by taking advantage of the subsumption relation that holds between the concepts. The top element of the lattice becomes the root of the hierarchy and each sequence of concepts in the lattice is associated with a path in the tree. This representation is also called a tree widget.

The tree representation has several advantages. As the metaphor of hierarchical folders is used for storing and retrieving files, bookmarks, menus items, etc., most users are familiar with it and hence no training on the part of the user is required. Furthermore, it takes little space on the screen and it may be drawn efficiently. The main disadvantage, as seen in the example above, is that there may be a considerable amount of duplication of information when the concepts have multiple parents. On the other hand, this is not very likely to happen if only some levels of the hierarchy are visualized, as frequently happens in many applications.

Nested line diagrams

Sometimes it is helpful to split a complex visualization into more individually comprehensible, multiple displays. This is the main rationale behind the technique now described, called nested line diagrams. It consists of four main steps:

1. Partition the set of attributes describing a given context in two subsets (for the sake of simplicity we refer to only two subsets, the generalization to more subsets being straightforward).

2. Find concept lattices L_l and L_2 of the subcontexts identified by the attribute subsets of step 1.

3. Place a copy of L_2 in each node of L_l.

4. Mark with filled circles the elements in each copy of L_2 that belong to the full lattice.

In other words, L_l is used as an outer frame in which to embed L_2, the copies of which act as a background structure. Mathematically, the full concept lattice is embedded in the direct product of the lattices of subcontexts as a join semilattice. As a result, the concepts and the edges of the full lattice are not directly represented in a nested line diagram. Instead, they can be derived by combining the information associated with the various levels of nesting.

Focus and context views

Unlike nested line diagrams, in which only one or more subsets of the original attributes are used at a time, the techniques discussed in this subsection consider the concept lattice to be built from the whole set of attributes. However, the lattice is not displayed in full, and not all displayed parts are rendered in the same way.

Greater prominence is given to a certain area of the visualization space where information of interest to the user is presumably located. In other terms, it is assumed that there is a current focus of interest in the lattice to be displayed. Such a focus usually corresponds to a certain concept, which may be identified in different ways depending on the application at hand.

The main feature of focus+context techniques is the smooth integration of the focus with the surrounding context. In contrast to other approaches based on multiple views such as local and global views, the disadvantage of which is that the user must map different graphical representations, the transition from focus to context is made more continuous by allowing for variable magnification of graphic displays.

In essence, the information contained in the graph is shown at varying levels of detail depending on the distance from the focus; the amount of information at the focal point is increased, whereas information placed further away is reduced in quantity.

3 Matrix decompositions

Matrix decompositions have been used for almost a century for data analysis and a large set of different decompositions are known. This text is quoted from book Understanding Complex datasets: data mining with matrix decompositions[37].

Recall that we consider a dataset as a matrix, with n rows, each of which represents an object, and m columns, each of which represents an attribute. The ijth entry of a dataset matrix is the value of attribute j for object i. Each family of matrix decompositions is a way of expressing a dataset matrix A, as the product of a set of new matrices, usually simpler in some way, that shed light on the structures or relationships implicit in A. Different matrix decompositions reveal different kinds of underlying structure.

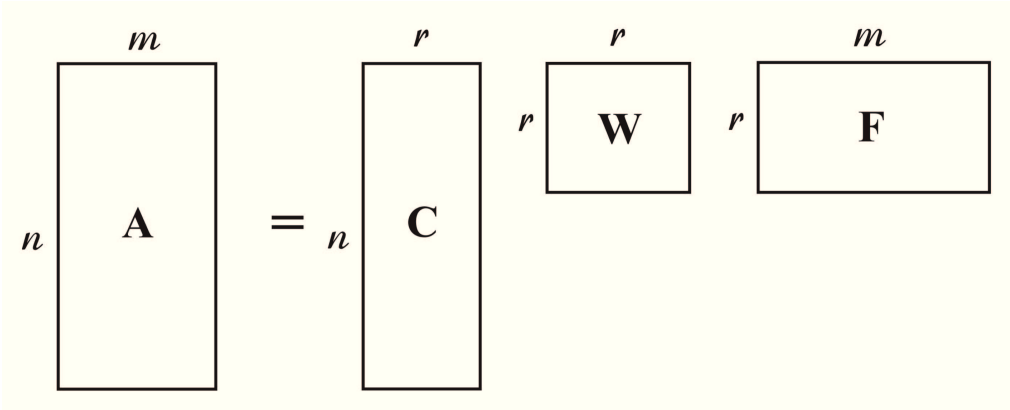


Figure 3.1: A basic matrix decomposition

More formally, a matrix decomposition can be described by an equation of this form

$$A = CWF]$$

where the sizes of the matrices are as follows:

A is $n \times m$; C is $n \times r$ for some r that is usually smaller than m; W is $r \times r$, and F is $r \times m$; figure 3.1 illustrates a matrix decomposition.

From this equation, an element of A, say a_{11}, arises from the multiplication of the first row of C, the top left element of W, and the first column of F, as shown infFigure 3.2. If we think of the rows of F as parts or pieces, then the product

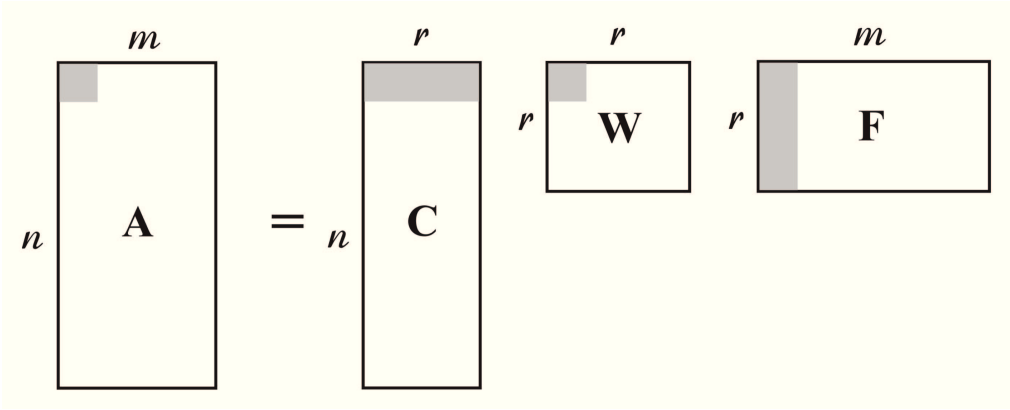


Figure 3.2: Each element of A is expressed as a product of a row of

WF weights each of the rows by the corresponding diagonal element of W. The matrix C then takes something from each part and combines them in a weighted way. Hence each entry of A is a kind of combination of parts from F, combined in ways described by C and W.

The matrix C has the same number of rows as A. Each row of C gives a different view of the object described by the corresponding row of A. In other words, the ith row of C provides r pieces of information that together are a new view of the ith object; while A provides m pieces of information about the ith object.

The matrix F has the same number of columns as A. Each column of F gives a different view of the attribute described by the corresponding column of A, in terms of r pieces of information, rather than the n pieces of information in A.

The role of r is to force a representation for the data that is more compact than its original form. Choosing $r = m$ still gives a sensible decomposition, but it is usually the case that r is chosen to be smaller than m. We are implicitly assuming that a more compact representation will capture underlying or latent regularities in the data that might be obscured by the form in which the data is found in A, usually because A expresses the data in a way that contains redundancies.

The particular dataset A, being studied is always considered to be a sample from a larger set of data that could have been collected. The use of a limited representational form prevents the matrix decomposition from over fitting the data, that is learning the precise properties of this particular dataset, rather than the properties of the larger system from which it came.

The matrix W has entries that reflect connections among the different latent or implicit regularities or latent factors - the ijth entry provides the relationship between the latent factor captured by the ith column of C (a kind of latent attribute) and latent factor captured by the jth row of F (a kind of latent object). For us, W will always be a diagonal matrix (that is, its off-diagonal elements are zero), in which case the latent factors for the objects and attributes are the same, and each entry can be interpreted as providing information about the relative importance of each underlying factor. Some decompositions do not create this middle

matrix, but we can always imagine that it is there as the $r \times r$ identity matrix.

We will consider many different kinds of matrix decompositions. These differ from each other in the assumptions they make about the kind of underlying structure that can be present in the data. In practice, this means that different matrix decompositions have different requirements for the entries of the matrices into which the dataset is decomposed, different relationships among the rows and columns, and different algorithms to compute each decomposition. Nevertheless, there are deep connections among the matrix decompositions we will consider. Most can be expressed as constrained optimization problems; and all are a form of Expectation-Maximization with stringent requirements on the distributions assumed[32].

3.1 Symmetry between objects and attributes

There is always a kind of symmetry between the objects and the attributes in a dataset because the matrix decomposition on the objects, as we have described it, can also be turned into a matrix decomposition on the attributes. For if

$$A = CWF$$

then

$$A' = F'W'C'$$

The dash indicates the transpose of the matrix, that is the matrix obtained by flipping the original matrix across its main diagonal, making rows into columns, and columns into rows. A' reverses the roles of objects and attributes, so the attributes are now the rows. On the right-hand side, F' plays the role originally played by C, and C' plays the role originally played by F.

For any matrix decomposition, whatever can be done with the objects in the dataset can also be done with the attributes, and vice versa[33].

3.2 Normalization

Because matrix decompositions are numerical computations, the magnitudes of the values in different columns (different attributes) must be comparable, or else the large magnitudes will have a greater influence on the result than the smaller ones. However, requires some care because it amounts to making assumptions, perhaps quite strong ones, about the data. Although matrix decompositions are usually characterized as non-parametric methods, the choice of normalization is really a parameter.

One standard way to adjust attribute values is to subtract the mean from the entries in each column, which centers the values around zero; and then divide each entry in each column by the standard deviation of the column mean. This

makes the values in different columns roughly similar in magnitude, but implicitly assumes that the values of each attribute are normally distributed.

When it is not clear how to normalize values in the dataset, as for example when the distribution of values is very different for different attributes, it can often be useful to replace the values in each column by their ranks. A common way to do this is to use the Spearman rank. The values are numbered in increasing order, except that when there are ties, the rank associated with the tied elements is the average of the ranks that those elements would have had if they had been different.

3.3 Degenerate decompositions

Many decompositions, in their simple forms, can be degenerate. Given an invertible $m \times m$ matrix X, it is often possible to insert XX^{-1} in the right hand side of a decomposition, rearrange, and get a new right-hand side that is another example of the same decomposition.

if

$$A = CF$$

then

$$A = C(XX^{-1})F = (CX)(X^{-1}F)$$

The parenthesized terms on the right-hand side are a new C and a new F, and so a different decomposition of A. Most matrix decompositions impose some further condition to specify which, of all these related decompositions, is 'the' decomposition[34].

3.4 Correlation matrices

Given a matrix A, we can form the matrices AA' and $A'A$, where the dash indicates the transpose of A. The matrix AA' is the correlation matrix of the objects. The magnitude of the ijth entry indicates the amount of correlation between the ith and the jth object. Similarly, the matrix $A'A$ is the correlation matrix of the attributes, and its entries indicate the amount of correlation between pairs of attributes. Both of these matrices are symmetric.

The correlation matrices can also be decomposed, and the resulting matrices analyzed to give new insights into the structures present in the data. However, this is often not as helpful as it seems, for three reasons. First, the correlation matrices are $n \times n$ and $m \times m$ respectively, so that at least the first can be very large. Second, calculating a decomposition for such a matrix can often be difficult because of numerical instability. Third, each decomposition of a correlation matrix provides information about the structure of the objects or about the structure of

the attributes, but not both at once. This can lose information implicit in their interactions.

3.5 Similarity and clustering

3.5.1 Geometric clustering

The rows of C can provide a clearer view of the properties of objects than the rows of A. Any data-mining clustering technique can be applied to the data as described by the rows of C and we might expect that the result will be a better clustering than a direct clustering of the data. However, this process assumes that the entries of C can properly be treated as coordinates, and that distances behave as expected. If the axes corresponding to the r rows of F are, for example, not orthogonal, then these assumptions are not correct. This does not mean that clustering will not be effective, but it should be done with some caution and with awareness of F.

3.5.2 Decomposition-based clustering; similarity clustering

All clustering depends on some measure of similarity between objects, or between attributes. We have seen that different interpretations correspond to different views of such measures: the geometric view corresponds to a metric such as Euclidean distance, while the component view corresponds to the element wise difference.

These interpretations therefore provide either hints or methods for clustering that exploit properties of the decomposition. For example, suppose that a dataset about customers contains details of their purchasing behavior, but also a customer number. It would be silly to use differences in this customer number as part of a distance measure. On the other hand, it might not be sensible to discard it from the dataset, since customer numbers are usually allocated sequentially in time, and this temporal information often has some predictive power.

3.5.3 Graph-based clustering

The similarities among objects in a geometric model are qualitatively different from the similarities in a graph model. For some datasets, it may be more suitable to cluster based on a pair wise affinity relationship between objects than to cluster geometrically.

The difference between the two views is the difference between a global view of similarity and a local view of similarity. In a geometric model, the distance between any two objects can be computed, and it stays the same regardless of

whether other objects are present or not. On the other hand, in a graph model, the distance between any two objects depends on which other objects are present and how they are arranged because the distance depends on a path or paths involving all of these objects.

In a larger sense, a geometric space has an existence on its own, and does not depend on the presence of objects. The shape of a graph space is not like this at all, all of the distances can be changed by the addition or removal of a single object. This suggests that care is needed with techniques that try to embed a graph space into a geometric one.

3.6 Finding local relationships

Although matrix decompositions do not look for local patterns in data in the same way as association rules, they can still be used to look more deeply at certain parts of the data. All rows of the dataset matrix are treated equally by a matrix decomposition but, as it is a numerical technique, it can be guided by changing the magnitudes of some rows compared to others. For example, if the entries in a row of the matrix are multiplied by two, then this will change the decomposition in a way whose effect is to consider that row as more important than the other rows (twice as important, in fact).

If we know that an object, or for that matter an attribute, is more important, then this information can be conveyed indirectly to the matrix decomposition using multiplication of a row or column by a scalar greater than one. In the same way, the effect of a row or column can be discounted by multiplying it by a scalar less than one.

This technique can be used to check whether a group of objects or attributes really are similar to each other, and to decide which of them might make a good pathfinder. Increasing the importance of one member of the group should have the effect of increasing the importance of the other members of the group in a coupled and visible way in the resulting decomposition.

This technique can also be used to look for clusters that are completely contained within other clusters. Such hidden clusters may sometimes be detected directly by density-based clustering but, even when detected, it may be difficult to find their boundaries. Increasing the importance of one or more objects suspected to be in the subcluster can have the effect of moving the entire subcluster, relative to the cluster that overlaps it, and so making the subcluster easier to see.

3.7 Sparse representations

A matrix is called sparse if most of its entries are zero. A decomposition that results in either C or F being sparse is of interest from the point of view of both analysis and practicality.

If the ith row of the matrix C is sparse, it means that the representation of object i in the transformed space is a particularly simple combination of the underlying parts. Sparse representations for the objects are attractive because they increase our confidence that the set of factors captures deeper realities underlying the dataset, and they allow more comprehensible explanations for the data. For example, some kinds of sparse independent component analysis seem to correspond to early-stage mammalian vision, where the input resources are well understood because they correspond to neurons. Sparse representations are also useful because they reduce the amount of space required to store representations of large datasets.

In the factor interpretation, a sparse row of C means that an object is made up of only a few of the factors. In the geometric interpretation, a sparse row means that each object has an extent in only a few dimensions. In the component interpretation, a sparse row means that each object merges values from only a few processes. In the graph interpretation, a sparse row means that paths from that object to the points corresponding to attributes pass through only a small number of way stations. These statements are all saying the same thing in a different way, but they once again illustrate the power of looking at properties of the data from different perspectives.

The F matrix can also be sparse. When this happens, it suggests that the parts are themselves particularly simple, requiring only a small amount of information, and that the parts are quite decoupled from each other.

We have pointed out that any matrix decomposition remains unchanged if the factor matrix is multiplied by an arbitrary invertible matrix, and the coordinate matrix is multiplied by its inverse. This corresponds to a rotation of the axes of the new space. It is sometimes useful to apply such a rotation at the end, after the decomposition has been computed, with the goal of making the representation more sparse. This reduces the optimality of the solution with respect to whatever criterion was used by the particular decomposition, but it may nevertheless increase the explanatory power of the result.

3.8 Algorithms and complexity

The matrices used for data analysis are often very large, so it is useful to have some sense of the complexity of computing matrix decompositions. Because matrix decompositions are numerical algorithms, it is also important to be aware of how numerical magnitudes affect results; this can sometimes cause computational problems such as instability, but can also be exploited to discover finer details of the structure present in the data.

Even looking at all of the elements of A has complexity $\Theta(nm)$, and it is hard to see how a useful matrix decomposition could avoid complexity $\Omega(nmr)$. In practice, most matrix decompositions are much more expensive, perhaps quadratic in one or both of n and m. Although quadratic complexity does not sound alarming, n can be extremely large so the execution time to compute a matrix decomposition

is often a limitation in practice.

Because n is often so large in real-world applications, matrix decompositions may not even be compute-bound. The performance bottleneck may actually be the time required to fetch the entries of A from the bottom of the memory hierarchy. This requires $\Theta(nm)$ operations, but the constants required in modern architectures are very large, typically comparable in size to m. Hence memory access times can be as bad as computation times.

For these reasons, there is a great deal of ongoing research aimed at exploiting sparse matrix algorithms; computing approximate matrix decompositions, for example low-rank approximations; and exploiting quantization of the matrix entries.

Many datasets have attributes that are categorical, that is they have values for which no natural ordering exists [23, 24, 25, 26, 27].

A lot of information about matrix decomposition are taken from more then one references like "Latent Semantic Indexing Via a SemiDiscrete Matrix Decomposition" [35], "Independent component analysis: Algorithms and applications" [36] and "Understanding Complex datasets: data mining with matrix decompositions" [37].

4 Singular Value Decomposition (SVD)

The singular value decomposition (SVD) transforms the data matrix in a way that exposes the amount of variation in the data relative to a set of latent features. The most natural interpretation is geometric: given a set of data in m-dimensional space, transform it to a new geometric space in which as much variation as possible is expressed along a new axis, as much variation independent of that is expressed along an axis orthogonal to the first, and so on. In particular, if the data is not inherently m-dimensional, its actual dimensionality the rank of the data matrix A is also exposed.

Singular value decomposition is well-known because of its application in information retrieval as LSI. SVD is especially suitable in its variant for sparse matrices [38][39][40].

This text is quoted from book Understanding Complex datasets: data mining with matrix decompositions[37].

The singular value decomposition of a matrix A with n rows and m columns is

$$A = USV'$$

where the superscript dash indicates the transpose of matrix V.

If A has rank r, that is r columns of A are linearly independent, then U is $n \times r$, S is an $r \times r$ diagonal matrix with non-negative, non-increasing entries $\sigma_1, \sigma_2, ..., \sigma_r$ (the singular values), and V' is $r \times m$. In addition, both U and V are orthogonal, so that $U'U = I$ and $V'V = I$. This is actually the so-called 'thin' SVD. If all of the singular values are different, the SVD is unique up to multiplication of a column of U and the matching row of V' by -1. In most practical datasets, $r = m$, since even if several attributes (that is, columns) are really measurements of the same thing, which is the commonest way in which the rank of A would be less than m, they are typically not exactly correlated.

By convention, the third matrix in the decomposition is written as a transpose. This emphasizes the duality between objects and attributes because both U and V are matrices whose rows correspond to objects and attributes respectively, and whose columns correspond to the r new parts. Unfortunately, this makes it easy to make mistakes about which way round V is considered, and which are its rows and columns.

The natural interpretation for an SVD is geometric, but the component interpretation is also useful [41].

4.1 SVD and Principal Component Analysis (PCA)

SVD is intimately connected with eigenvectors and eigenvalues. Principal component analysis (PCA) is another way to understand data, and there is considerable disagreement about the differences between the two techniques. Some authors consider them to be identical, others to differ in normalization strategies, and still others consider them to be completely distinct.

Theorem. (Singular Value Decomposition) Let A be an $m \times n$ with rank-r matrix. Be $\sigma_1, \sigma_2, ..., \sigma_r$ eigenvalues of a matrix $\sqrt{AA^T}$. There exist orthogonal matrices $U = (u_1, u_2, ..., u_r)$ and $V = (v_1, v_2, ..., vr)$, whose column vectors are orthonormal, and diagonal matrix $S = diag(\sigma_1, \sigma_2, ..., \sigma_r)$. The decomposition $A = USV^T$ is called singular value decomposition of matrix A and numbers $\sigma_1, \sigma_2, ..., \sigma_r$ are singular values of the matrix A. Columns of U (or V) are called left (or right) singular vectors of matrix A.

Now we have a decomposition of original matrix A. It is not needed to say, that the left and right singular vectors are not sparse. We have at most r nonzero singular numbers, where rank-r is the smaller of the two matrix dimensions. Because the singular values usually fall quickly, we can take only k greatest singular values and corresponding singular vector coordinates and create a k-reduced singular decomposition of matrix A.

Definition. Let us have $k, 0 < k < r$ and singular value decomposition of A

$$A = U_k S_k V_k^T$$

is called a k-reduced singular value decomposition (k-rank SVD).

Theorem. (Eckart-Young)[42]. Among all $m \times n$ matrices C of rank at most k, A_k is the one that minimizes

$$||A_k - A||_F^2 = \sum(A_{i,j} - C_{w,j})2$$

Because k-rank SVD is the best k-rank approximation of original matrix A, any other decomposition will increase the sum of squares of matrix $A - A_k$.

The SVD is hard to compute and once computed, it reflects only the decomposition of original matrix. The recalculation of the SVD is expensive, so it is impossible to recalculate the SVD every time new rows or columns are inserted.

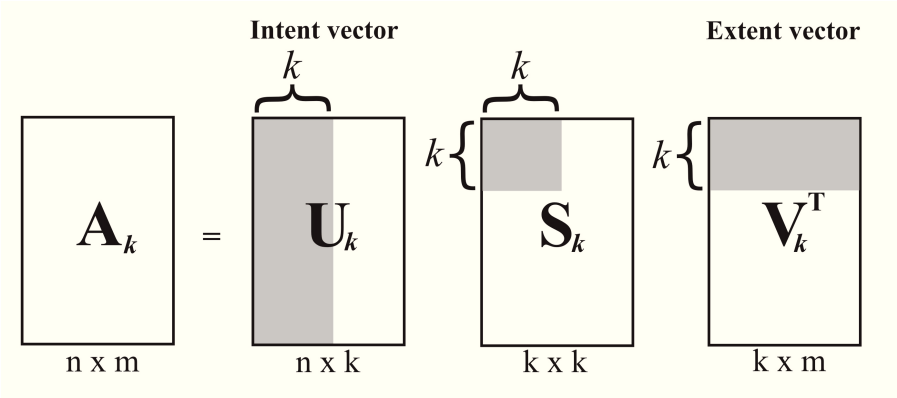


Figure 4.1: k-reduced singular value decomposition

SVD-Updating is a partial solution, but since the error slightly increases with inserted rows and columns, if the updates happen frequently, the recalculation of the SVD may be needed soon or later [43].

Most versions of principal component analysis find the eigenvectors and eigenvectors of either the matrix AA', which describes the correlation among the objects, or the matrix $A'A$, which describes the correlation among the attributes.

PCA is limited in at least the following two ways: first, it analyzes either the objects or the attributes independently, whereas SVD analyzes both together; and second, the correlation matrices are expensive to form (AA' is $n \times n$ which makes it difficult to handle) and often ill-conditioned, for the reason that computing the eigenvectors is problematic.

4.2 Normalization

Because SVD is a numerical algorithm, it is important to ensure that the magnitudes of the entries in the dataset matrix are appropriate, so that properties are compared in a way that accords with comparisons in the real world.

For example, height and weight are roughly correlated in humans. However, if height is measured in miles, and weight in grams, then weight is going to seem much more important during the decomposition.

In general we don't know what the 'right' units are for each attribute. In the absence of better information, the only sensible thing to do is to scale all of the attribute values into roughly the same range. This encodes an assumption that all attributes are of about the same importance. This is quite a strong assumption, but it is hard to see how to do better.

If the values in the data matrix A are all positive, the first component of the decomposition will capture the rather trivial variation along the axis that joins the origin to the centered of the data (in m-dimensional space). We could, of course, ignore this component in subsequent analysis. The problem is that the

new axes are forced to be orthogonal to each other, so that the second axis points in a distorted direction. This is illustrates in figure 4.1, where the top ellipse shows what happens when positive data is transformed. The second axis does not properly capture variation in the data because of the existence of the first axis. The bottom ellipse shows what happens when the data is zero centered, that is, for each column, the mean of that column is subtracted from each entry. This moves the data 'cloud' so that it is centered at the origin. Now the new axes correctly capture the directions of variation in the data [44].

4.3 Interpreting an SVD

The geometric interpretation is most natural for an SVD, there is something to be learned from the other interpretations.

4.3.1 Factor interpretation

Interpreting the rows of V' (the columns of V) as underlying factors is perhaps the oldest way of understanding an SVD. For example, suppose we want to understand what makes people happy. We might suspect that factors such as income, education, family life, marital status, and a satisfying job might all be relevant, but we couldn't be sure that these were all the factors, and we might not be sure precisely how to measure them. Designing a questionnaire to ask about such factors, and also about degree of happiness, might need questions directly about income, but also questions about home ownership, pension plan, and medical insurance. It might turn out that all of these correlate strongly with income, but it might not, and the differences in correlation may provide insight into the contribution of a more general concept such as 'prosperity' to happiness. The survey data can be put into a matrix with one row for each respondent, and one column for the response each question.

An SVD of this matrix can help to find the latent factors behind the explicit factors that each question and response is addressing.

For datasets of modest size, where the attributes exhibit strong correlations, this can work well. For example, figure 4.2 is derived from a dataset in which 78 people were asked to rank 14 wines, from 1 to 14, although many did not carry out a strict ranking. So the attributes in this dataset are wines, and the entries are indications of how much each wine was liked by each person[45].

The figure shows a plot along the first two axes of the transformed space, corresponding to the two most important factors. Some further analysis is required, but the first factor turns out to be liking for wine - those respondents at the left end of the plot are those who like wine, that is who had many low numbers in their 'ranking', while those at the right end liked wine less across the board. This factor corresponds to something which could have been seen in the data relatively easily

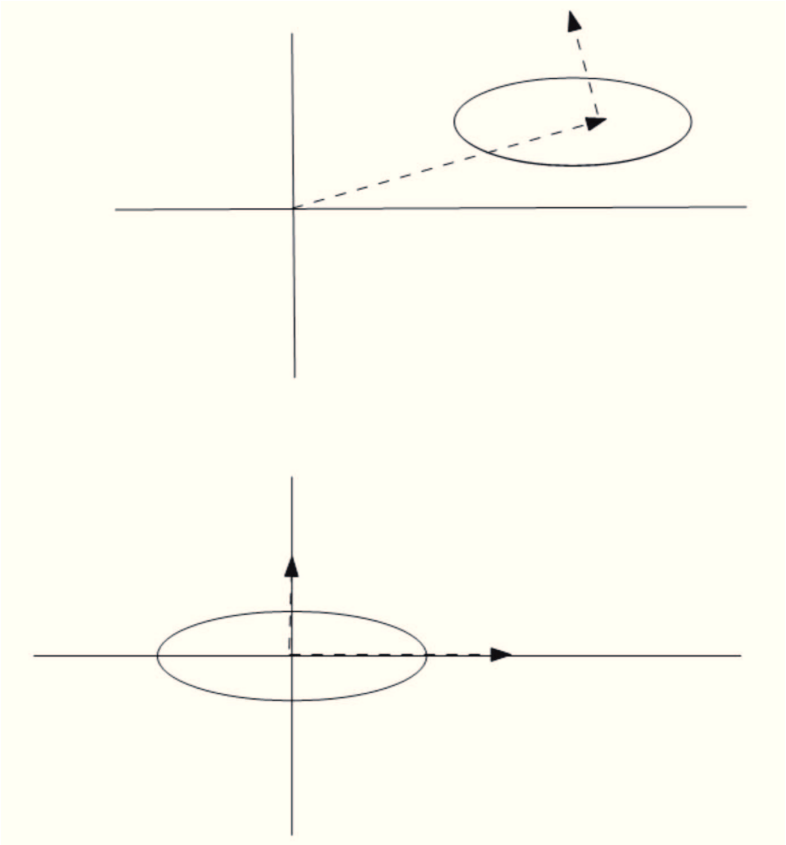


Figure 4.2: The first two new axes when the data values are positive (top) and zero-centered (bottom)

since it correlates strongly with the sum of the 'rankings'. For example, the outlier at the right end corresponds to someone who rated every wine 14.

The second factor turns out to indicate preference for red versus white wine - those respondents at the top of the plot prefer red wine over white, while those at the bottom of the plot prefer white over red. This factor is much less easy to see directly in the data. Notice also that the matrix decomposition does not know the 'meaning' of any column of the dataset. It discovers this pattern by noticing that certain rankings are correlated with other rankings.

One obvious conclusion that can be drawn just from seeing the triangular shape of the plot in the figure is that those who like wine a lot do not have strong preferences for red versus white; it is those who like wine less who tend to have such a preference. These simple results have immediate implications for marketing wine.

However, for large complex datasets, the factors tend to be linear combinations of all or most of the attributes in the dataset because each attribute is partially correlated with many of the others in subtle ways. Hence, it is often difficult to interpret the factors and relate them to the application domain, from

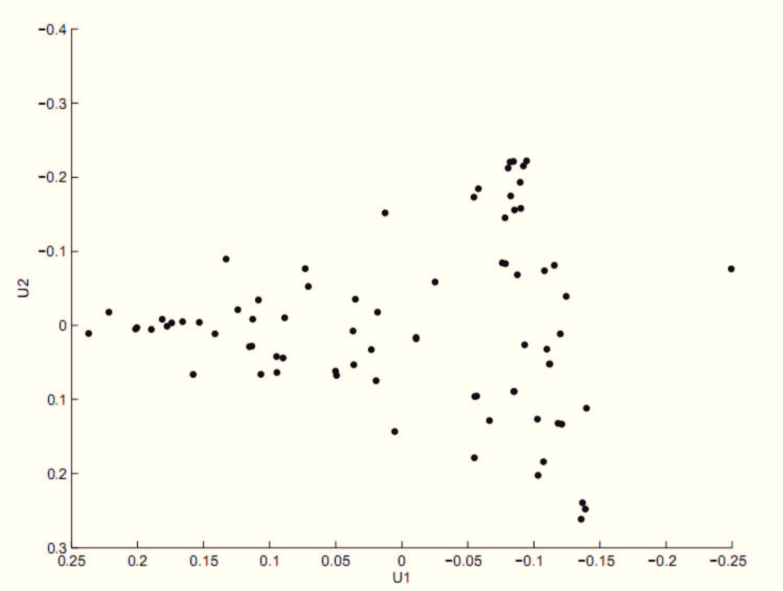


Figure 4.3: The first two factors for a dataset ranking wines

which the original attributes come[46].

4.3.2 Geometric interpretation

The geometric interpretation of an SVD is to regard the rows of V (columns of V_1) as defining new axes, the rows of U as coordinates of the objects in the space spanned by these new axes, and S as a scaling factor indicating the relative importance (or stretching) of each new axis. Note that the possible non-uniqueness of the decomposition means that an axis can be flipped without changing anything fundamental.

Because the SVD is symmetric with respect to rows and columns, it can also be regarded as defining a new space spanned by the rows of U and mapping the attributes from coordinates in an original n-dimensional space into this new space. The maximum variation among the attributes is captured in the first dimension, and so on.

The most useful property of the SVD is that the axes in the new space, which represent new pseudo attributes, are orthogonal. Hence the explicit properties of each object as characterized by the original attributes are expressed in terms of new attributes that are independent of each other. As we saw in figure 4.2, the orthogonality of the new axes means that the rows of the C matrix can be plotted in space in a way that accurately reflects their relationships [46].

4.3.3 Rotation and stretching

There are several intuitive ways to understand how an SVD is transforming the original data.

First, notice that when we interpret the rows of A as coordinates in an m-dimensional space, the axes of this space can be made explicit by writing A as AI, where I is the m-dimensional identity matrix. These axes are just the ordinary Cartesian axes. The matrix decomposition is asserting the equivalence of these coordinates and ordinary axes to new coordinates (the rows of U) and a new set of axes described by the product SV'. The matrix V' is a rotation, and the matrix S is a stretching of the axes. However, these axes are not arbitrary; rather they have been computed based on the data itself.

Imagine the unit sphere in m dimensions. Then V' followed by S rotates and stretches this unit sphere so that it fits 'over' the data. This fitting guarantees that the coordinates required to describe each object will be as simple as possible. figure 4.3 illustrates the process. The gray ellipse represents the raw data. First the axes are rotated to align with the axes of the rough ellipse formed by the data. Then the axes are stretched so that they better fit the extents of the data. Relative to these new axes, the coordinates of each data point are simpler [46].

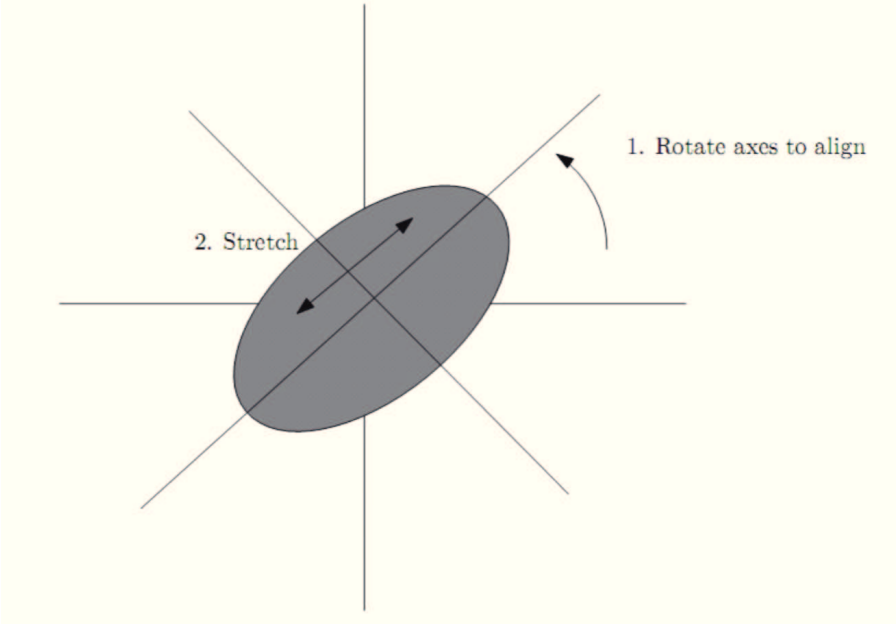


Figure 4.4: One intuition about SVD: rotating and scaling the axes

This intuition also shows clearly the effect of normalization on the SVD. Zero centering places the rough ellipse corresponding to the data close to the origin. Dividing the entries in each column by their standard deviation from the mean makes the structure of the data as close to a sphere as possible - so that the rotation and scaling can concentrate on the distribution or density of the objects in each direction.

A special case that can also be understood from this intuitive point of view

is when the raw data appears to have dimensionality m but is actually a lower-dimensional manifold. This situation is shown in figure 4.4. Here the raw data seems to have dimension two - each data point requires an x and y coordinate to describe its position (or the value of two attributes). However, the rotation shows that only one new axis and one stretch factor are required to fully describe the dataset.

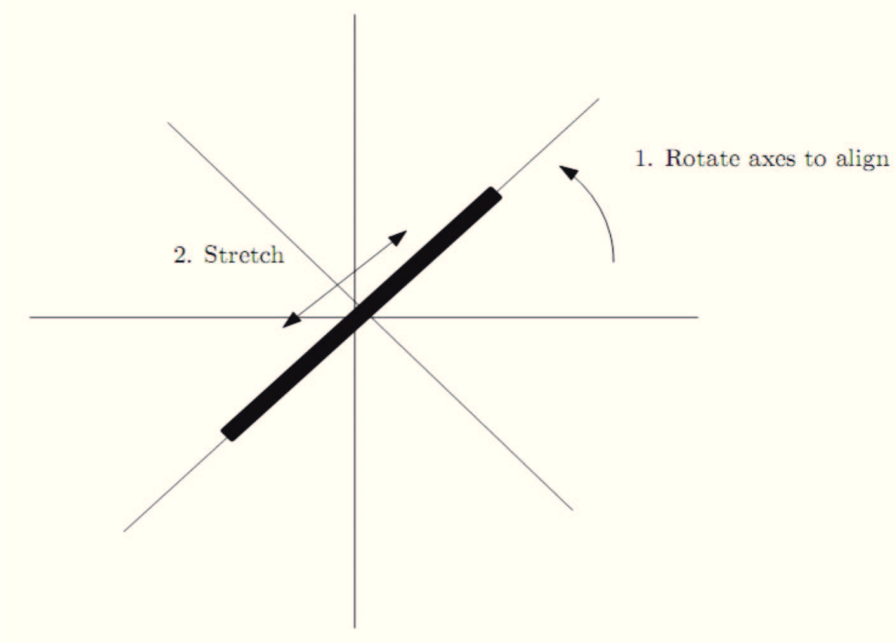


Figure 4.5: Data appears 2-dimensional but can be seen to be 1-dimensional after rotation

4.3.4 Springs

Another helpful way to think about the transformation that SVD does is the following. Suppose we place points corresponding to both the objects and the attributes in the same m-dimensional space. Now connect the ith object to the jth attribute by a spring whose tension corresponds to the magnitude of the ijth entry of the matrix A. (If the entry is negative, then the spring is repulsive rather than attractive.) Then the stable positions where each point is at rest correspond to the locations described by the U matrix, for objects, and the V matrix, for attributes. Actually, to get the scaling right, these locations correspond to locations described by $US^{1/2}$ and $VS^{1/2}$ because the singular values describe the scaling of each dimension relative to the others.

This view of SVD illustrates the symmetry between objects and attributes. It also shows how SVD makes use of the indirect and higher-order relationships among object and attributes. If we assume that all entries of the matrix are non-zero, then the relative position of two objects depends on the positions of all of the

attributes; but these, in turn, depend on the position of these two objects, but also of all of the other objects. The position of these objects depends on the strength of their connection to the attributes, and so on. The SVD represents the fixed point of this reasoning (the stable positions into which the points settle) and so takes into account all of the correlation information between objects and attributes.

4.3.5 Component interpretation

Let u_i be the ith column of U, s_i the ith singular value of S and v_i the ith row of V. Then

$$A = \sum_{i=1}^{m} A_i$$

where $A_i = u_i s_i v_i^{'}$. This sum tells us that we can think of each entry of A as the sum of the corresponding entries in each of the A_i, and of A as the point wise sum of the A_i. In other words, the A_i form layers that together recreate A.

This view is exactly what we hypothesized was true for many real-world datasets: the value of a particular entry in the dataset is the result of the superposition of a number of processes, only some of which are of interest. For SVD, the layers represent independently varying values [46].

Of course, there is no necessary reason why the decomposition into layers that an SVD provides should correspond to the set of underlying processes that were at work when the dataset was collected, but a correspondence can often be found in practice, at least in the earlier dimensions.

4.3.6 Graph interpretation

The graph interpretation of SVD takes a bipartite graph, whose two kinds of objects correspond to objects and to attributes, and whose edges are weighted by the entries of the matrix, and expands it to a tripartite graph. In this tripartite graph, there is a third set of r way station vertices corresponding to the 'middle' dimension of the SVD. The vertices corresponding to the objects are fully connected to the way station vertices that are created by the decomposition; and these in turn are fully connected to the vertices corresponding to the attributes.

Each edge in the tripartite graph has an associated weight. Those connecting objects to way stations get their weights from the entries of the matrix $US^{1/2}$, and those connecting way stations to attributes get their weights from the entries of the matrix $VS^{1/2}$. The fact that the product of these matrices is A means that these weights fit together properly. The sum of the weights along all of the paths between a particular object i and an attribute j is the ijth entry of A, as long as the weights along a path are accumulated by multiplication. These weights can be understood as capturing the similarity between the vertices they connect; or

equivalently the permeability of the connection between them, or how easy it is to travel from one end to the other.

The intuition here is that an SVD allocates capacity to each edge to optimize the total permeability of all paths. The weight associated with an edge from, say, an object to a way station must be assigned so that it fits with the paths from that object to all of the attributes, since this path makes a contribution to all of them.

4.4 Applying SVD

4.4.1 Selecting factors, dimensions, components, and way stations

The main distinguishing feature of an SVD is that it concentrates variation into early dimensions. This means that the natural way to select parts of the structure inside the dataset is to select, from the r components, the first k.

We have suggested that there are two main reasons to select and retain only some parts of a decomposition: because the discarded parts are considered noise; or because the discarded parts represent some process that we do not wish to model. Given the ordering of the parts by an SVD, these decisions are much the same. The only difference is that we might use slightly different criteria to choose how many parts to retain and how many to discard.

Suppose that we want to represent the dataset properties in a space of dimension k (where $k \leq r$), that is we want to retain only k parts of the decomposition. The first k rows of V' define the axes of a k-dimensional space. Surprisingly good representations of spaces with many hundreds of dimensions can be achieved by quite small values of k, perhaps less than 10.

4.4.2 Selecting objects and/or attributes with special properties

The correlation matrices AA' (for the objects) and $A'A$ (for the attributes) provide information about the relationships in the dataset. However, the equivalent truncated correlation matrices provide even better information, and in a way that can be related to the SVD [47,48].

Let A_k be a matrix obtained by multiplying together some k rows of U, the matching elements of S, and the matching rows of V. Consider the correlation matrix $A_k A_k^{'}$, which we might expect to tell us something about the correlation among objects due to the k subprocess(es) that remain. Expanding A_k using the SVD we find that

$$A_k A_k^{'} = (U_k S_k V_k^{'})(U_k S_k V_k^{'})^{'}$$

$$= U_k S_k V_k^{'} V_k S_k U_k^{'}$$

$$= U_k S_k^2 U_k^{'}$$

since $V_k^{'} V_k = I$ and $S_k^{'} = S_k$. So the ijth entry of $A_k A_k^{'}$ is the dot product of the ith and jth rows of U_k, weighted by the squares of the singular values. (In exactly the same way, the entries of $A_k A_k^{'}$ are weighted dot products of the rows of V_k).

The magnitudes of the entries in the correlation matrix obtained by truncating after the first few singular values provide a good estimate of the correlation between objects and/or attributes for the process represented by the choice of k.

Unlike the direct correlation matrix of A, the correlation matrix after truncation reflects both the absence of correlations in the processes that have been ignored (the lost information due to truncation), and higher-order truncation information. For example, two objects with no direct correlation may have indirect correlations via some other object, and their mutual correlations with some of the attributes. Matrices such as $A_k A_k^{'}$ may be useful inputs to other analysis techniques since they encapsulate information neatly.

4.4.3 Denoising

A dataset that contains noise may appear to be of much higher dimensionality than it really is. figure 4.6 shows, as a dark line, a 1-dimensional dataset with two (perfectly correlated) attributes, and so appearing 2-dimensional. SVD will quickly detect that the data is actually 1-dimensional. The dashed ellipse shows what happens when noise is added to the dataset. The data now has an apparent extent in the second direction.

After the SVD transformation the data will appear to be 2-dimensional - but the extent, and so the amount of stretching, required in the second dimension will be small. This is the clue that this dimension does not contain real structure. Because an SVD arranges the dimensions in decreasing order of the magnitude of the singular values, the later dimensions with little or no structure will appear at the end.

An appropriate choice for k is made by considering the magnitude of the singular values, which provide a measure of how much variation is being captured in each dimension. There are two standard ways to make this choice, and at least two other, more sophisticated, methods that are beginning to be used.

The first standard approach is to plot the singular values using a scree plot, a plot of the magnitudes of the singular values in order. Since these are non-increasing, and often decrease quite quickly, they resemble the side of a mountain, which is the origin of the name. A suitable cutoff is a value of k where this slope seems to flatten or when there is a detectable elbow or knee in the curve.

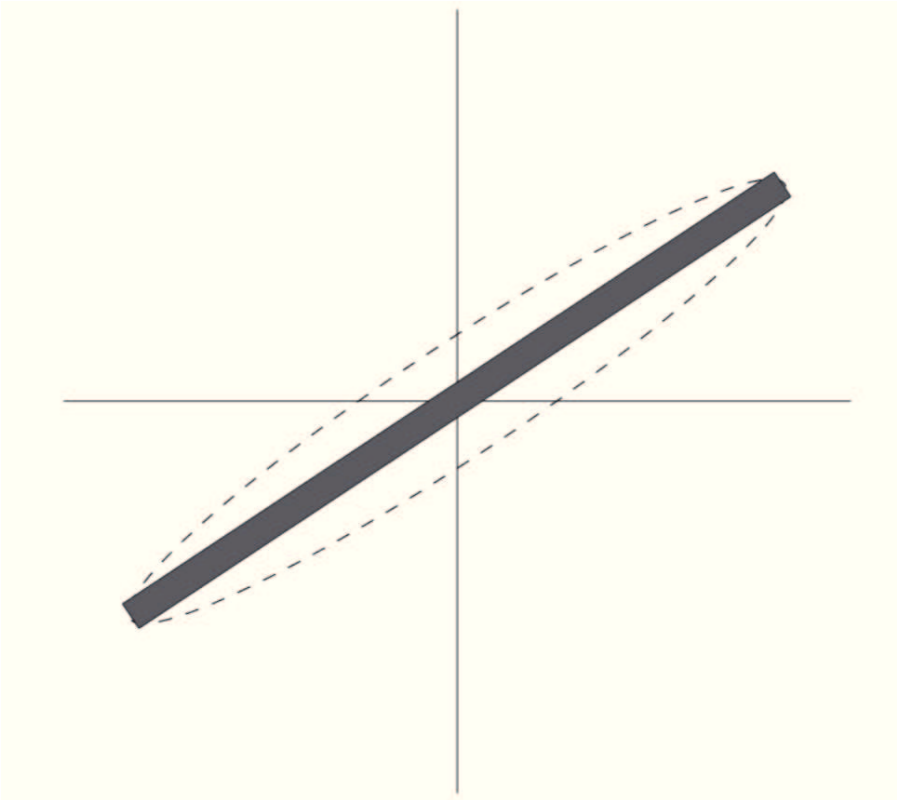


Figure 4.6: The effect of noise on the dimensionality of a dataset

The second considers the contribution of each singular value to the whole in a slightly more formal way. The contribution of each singular value can be computed by

$$f_k = S_k^2 / \sum_{i=1}^{r} S_i^2$$

and then the entropy of the dataset calculated as

$$entropy = \frac{-1}{\log_r} \sum_{k=1}^{r} f_k \log(f_k)$$

Entropy measures the amount of disorder in a set of objects; in this case, it has a value between 0 (all variation is captured in the first dimension) and 1 (all dimensions are equally important) [49]. The magnitude of the entropy indicates how many dimensions need to be retained.

The third method is to use the technique of Zhu and Ghodsi [50] which is based on the assumption that the singular values are drawn from two different distributions, one for the significant components and the other for the noise components. An expression for the profile log-likelihood of the choice of k is constructed from the combination of these distributions, and the maximum log-likelihood is

determined empirically. This maximum corresponds to the best choice of k.

A fourth method is to choose k such that the residual matrix of the k +1 to m components appears to be a random matrix. Suppose a matrix is multiplied pointwise by a random -1,+1 matrix. Its Frobenius norm does not change. If it is a random matrix, that is it contains no structure, only noise, its 2-norm will not change either. However, if it contains structure, altering the signs of its entries will change the 2-norm by an amount that reflects the amount of structure present. Hence the difference of the 2-norms of the residual matrix and the matrix obtained from it by pointwise multiplication by a random -1,+1 matrix, divided by the Frobenius norm should become small as soon as the residual matrix contains only noise [51].

The truncated SVD is the best representation of the data in the sense of capturing the variation among the objects and attributes. The matrix A_k that results from remultiplying the truncated matrices on the right hand side of the decomposition is the best approximation to A in both the 2-norm and Frobenius norm. A truncated SVD is the best minimum variance estimation of the random variable corresponding to the rows; in fact truncation corresponds to minimum variance estimation. Hence an SVD provides the best representation of the data in a statistical sense as well[52].

4.4.4 Similarity and clustering

The main advantage of an SVD is that, under the geometric interpretation, truncating the U and V matrices avoids the difficulties of working with metrics in high-dimensional spaces, while preserving as accurate a representation in low dimension as possible.

The two commonest measures of similarity among objects or attributes are:

- Euclidean distance. Computing the Euclidean distance between a pair of points is cheaper and more effective than computing the distance between them in the original space.
- Cosine similarity. This measures the closeness of the two vectors from the origin to each of the points and is useful when the appropriate rows or columns of A have been normalized so that the points are effectively on the surface of a unit sphere; or when the entries in the matrix are sparse and sparsity would be destroyed by normalization. This happens, for example, in word-document matrices which are sparse because most words occur in only a few documents, and where the fact that a word occurs at all in a document is more interesting than its frequency.

A vast number of clustering techniques based on SVD have been developed. Often this has happened in particular problem domains, and many of them are not well known outside of these domains. Several have been reinvented repeatedly.

They all rely on taking some subset of the singular vectors: the columns of U for clustering objects and the columns of V for clustering attributes.

Here are some techniques for clustering:

- Use an ordinary clustering technique on the singular vectors, for example k-means. Applying such a technique to the rows of U (especially after truncation) rather than the rows of A exploits the fact that 'noise' has been removed from the dataset and its dimensionality reduced. So the ordinary clustering technique should produce a better result and also run faster.
- Treat the new axes (the right singular vectors) as cluster centroids and allocate each object to the appropriate cluster, giving priority to the right singular vectors in order (and the converse for clustering attributes). In other words, the first cluster contains all of those points that fall within a 45° cone around the first new axis, the second cluster contains all points that fall within a cone around the second new axis, and so on. The points that fall in the cone around the kth new axis can be treated as a cluster in the same way as the others, or could be considered as the 'everything else' cluster, in other words as a set of outliers. Each cone is really two cones, one consisting of vectors positively correlated with the axis, and the other consisting of vectors negatively correlated with it. In some applications, it might be sensible to consider objects in both cones as forming a single cluster; in others they might be considered as forming two different clusters.
- Look for 'plateaus' in the first left singular vectors: either by sorting the values from a column of U (usually the first column) and plotting them directly [53], or by histogramming their values. This approach has considerable theoretical support, but it is hard to use in practice because (a) clear plateaus and steps between them do not tend to appear in real datasets, and (b) in any case the boundaries between such structures tend to require subjective choice[54].

There are also ways to consider the entries in a matrix as defining the edges of a graph, and then partitioning this graph to cluster the data.

4.4.5 Finding local relationships

Normalization of datasets for an SVD means that their data values are of similar magnitude. Multiplying row(s) or column(s) of the dataset by a scalar effectively changes their influence on the entire decomposition. If the scalar is greater than one, the effect is to move the points corresponding to these rows or columns further from the origin. However, this also has the useful side-effect of 'pulling' points that are correlated with the upweighted points further from the

origin as well. Furthermore, increasing the weight on some objects also moves the points corresponding to attributes that are correlated with these objects.

This property can be used to see clusters of correlated objects and attributes that would otherwise be hidden inside larger clusters, provided at least one such object is known. If this known object is upweighted, then it will move, and also change the position of correlated objects. All of these objects can then be upweighted and the SVD repeated. When new points stop being moved outwards, the current set of upweighted objects probably represents a well-defined cluster.

The same process can be used to determine roughly which attributes account for membership of a cluster of objects (and vice versa). For if increasing the weight on the objects in the cluster has the effect of moving some set of attributes, then increasing the weight on those attributes should have the effect of moving the cluster of objects - and this can be checked by the appropriate SVDs.

One of the weaknesses of SVD is that the pseudoattributes or dimensions of the transformed spaces cannot be easily understood because they are linear combinations of all of the original attributes. However, the significance of the first few dimensions can sometimes be discovered by adding extra artificial objects to the dataset representing extremal examples of some property of interest. For example, if we suspect that the first transformed dimension is capturing the total magnitude of the attributes associated with each object, then we can add artificial objects whose total magnitudes are larger than, and smaller than, those of any normal object in the dataset. If the points corresponding to the artificial objects are at the extremes of one dimension in the transformed space, then we can be confident that this dimension is capturing total magnitude. For example, recall the first dimension of the transformed wine dataset, with one person who had given all of the wines low scores.

Matrix decompositions are usually applied to datasets that do not have a target attribute. However, if a target attribute is known, or if we are interested in investigating how one particular attribute is affected by the others, SVD can provide some insight.

If the target attribute takes two values (a situation we can create for an arbitrary attribute by choosing a midpoint value), then the matrix can be divided into two parts: one associated with one value of the target attribute, and the other with the other value. The SVD of each of these matrices produces two V matrices, say V_1 and V_2. If the points from these matrices are plotted in the same space, then the different positions of each attribute in V_1 and V_2 give an indication of how the attribute interacts with the two values of the target attribute. Attributes that move a long distance from one plot to the other tend to be good predictors of the target attribute.

This technique implicitly assumes that the two submatrices are reasonable samples from some larger universe of data and so their SVDs can be plotted in the same space. This assumption may not be valid for particular datasets.

4.5 Algorithm issues

4.5.1 Algorithms and complexity

The complexity of SVD is

$$n^2m + nm^2 \qquad \text{(in [3])}$$

Since m is typically much smaller than n in data mining algorithms, the complexity is usually taken to be $O(n^2m)$. In data-mining applications, n is often large, so computing the SVD is expensive.

Many computational packages (for example, Matlab [119], Octave[120]) contain a command to compute an SVD. Standalone software in most programming languages is also readily available.

4.5.2 Updating an SVD

An SVD can be updated in two senses. The first is that the matrix A remains the same size, but has had some of its values changed. In this case, it is straightforward to recompute the SVD incrementally [55, 56]. The time complexity is linear in n provided that the magnitude of the changes are small.

An SVD can also be updated in the sense that new rows or columns are added. Rearranging the SVD equation we see that

$$U = AVS^{-1}$$

Hence given a new row of A, whose shape is $l \times m$, this equation can be applied to transform it to a new row of U, whose shape is $l \times r$. A similar procedure can be used to update V. This computation is not a true update since it does not preserve orthogonalities, but it is cheap. If desired, the previous incremental algorithm can be run on the new matrices to reinforce the orthogonality.

5 Non-Negative Matrix Factorization (NMF)

Non-negative matrix factorization is really a class of decompositions whose members are not necessarily closely related to each other. They share the property that they are designed for datasets in which the attribute values are never negative - and it does not make sense for the decomposition matrices to contain negative attribute values either. Such datasets have attributes that count things, or measure quantities, or measure intensities. For example, documents cannot contain negative occurrences of words; images cannot contain negative amounts of each color; chemical reactions cannot involve negative amounts of each reagent, and so on. This text is quoted from book Understanding Complex datasets: data mining with matrix decompositions[37].

A side-effect of this non-negativity property is that the mixing of components that we have seen is one way to understand decompositions can only be additive. In other words, a decomposition can only add together components, not subtract them. And the pieces themselves do not have any negative structure, so the combining really is additive - including a new component cannot decrease the size of any matrix entry. It is natural to think of the factors or components as parts that are put together additively.

The matrix decompositions we have seen so far will potentially decompose a non-negative matrix in such a way that either the factors or the mixing involve negative values. If there are negative values in the factor matrix, then the factors must somehow describe the absence of something. If there are negative values in the mixing matrix, then constructing the data matrix must require subtracting some components. In the kind of settings mentioned above, neither of these possibilities has a natural interpretation, so the non-negativity constraint seems appropriate (although it should be kept in mind that imaginary numbers have similar drawbacks, but have turned out to be useful in constructing solutions to a wide variety of problems).

Non-negative matrix factorization (NMF) was developed to address settings where negative values in the component matrices do not seem appropriate. One of the first efficient algorithms that computed an NMF, also had the property that the decomposition was sparse, that is each entry in the dataset matrix is expressed as the sum of a small number of factors; in other words, the mixing matrix contains many zeroes[57].

NMF (Non-negative Matrix Factorization) called also PMF (Positive Matrix Factorization) is an emerging technique for data mining, dimensionality re-diction,

pattern recognition, object detection, classification, gene clustering, sparse non-negative representation and coding, and blind source separation (BSS) [57, 58, 59, 60, 61, 62]. The NMF, first introduced by Paatero and Trapper, and further investigated by many researchers [63, 64, 65, 66], does not assume explicitly or implicitly sparseness, smoothness or mutual statistical independence of hidden (latent) components, however it usually provides quite a sparse decomposition [57, 61, 65]. NMF has already found a wide spectrum of applications in PET, spectroscopy, chemo metrics and environmental science where the matrices have clear physical meanings and some normalization or constraints are imposed on them (for example, the matrix A has columns normalized to unit length) [58, 59, 61, 63]. Recently, we have applied NMF with temporal smoothness and spatial constraints to improve the analysis of EEG data for early detection of Alzheimer's disease. A NMF approach is promising in many applications from engineering to neuroscience since it is designed to capture alternative structures inherent in data and possibly to provide more biological insight. Lee and Seung introduced NMF in its modern formulation as a method to decompose patterns or images [57].

Nonnegative matrix factorization differs from other rank reduction methods for vector space models in text mining by using constraints that produce nonnegative basis vectors, which makes the concept of a parts-based representation possible. Basis vectors contain no negative entries. This allows only additive combinations of the vectors to reproduce the original. The perception of the whole becomes a combination of its parts represented by these basis vectors. In text mining, the vectors represent or identify semantic features, i.e., a set of words denoting a particular concept or topic. If a document is viewed as a combination of basis vectors, then it can be categorized as belonging to the topic represented by its principal vector. Thus, NMF can be used to organize text collections into partitioned structures or clusters directly derived from the nonnegative factors.

The standard definition for non-negative matrix factorization (NMF) of the matrix A is

$$V = WH$$

where V is $n \times m$, W is $n \times k$ and H is $k \times m$, and $k \leq n$. Both W and H must contain only non-negative entries. W is the matrix of factors and H is the mixing matrix.

The NMF problem is defined as finding a low rank approximation of V in terms of some metric (e.g., the norm) by factoring V into the product (WH) of two reduced-dimensional matrices W and H. Each column of W is a basis vector, i.e., it contains an encoding of a semantic space or concept from V and each column of H contains an encoding of the linear combination of the basis vectors that approximates the corresponding column of V. Dimensions of W and H are $(n \times k)$ and $(k \times m)$ respectively, where k is the reduced rank or selected number of topics. Usually k is chosen to be much smaller than m, but more accurately, $k << min(n, m)$. Finding the appropriate value of k depends on the application and is also influenced by the nature of the collection itself.

Common approaches to NMF obtain an approximation of V by computing a (W, H) pair to minimize the Frobenius norm of the difference $V - WH$. Let $V \in R^{n \times m}$ be a nonnegative matrix and $W \in R^{n \times k}$ and $H \in R^{k \times m}$ for $0 < k << min(n, m)$. Then, the objective function or minimization problem can be stated as:

$$min||V - WH||_F^2$$

with $W_{ij} > 0$ and $H_{ij} > 0$ for each i and j.

The matrices W and H are not unique. Usually H is initialized to zero and W to a randomly generated matrix where each $W_{ij} > 0$ and these initial estimates are improved or updated with alternating iterations of the algorithm. In the following subsections some existing NMF techniques are discussed[67].

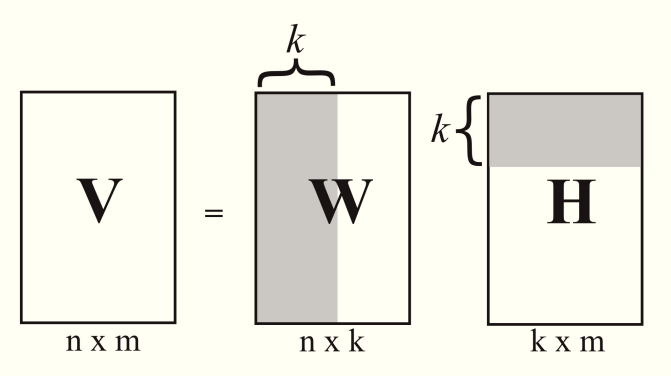


Figure 5.1: k-reduced Nonnegative Matrix Factorization (NMF)

5.1 Factor interpretation

The natural way to interpret an NMF is as defining a set of factors, and a mixing of those factors to produce the observed data. Because of the non-negativity, the factors can be interpreted as parts, and the mixing as addition of parts. In both ways, NMF has attractive simplicity. The factor interpretation has been successful when the underlying data are images or signals.

However, this is not automatically the case, and the factors produced are not always easy to interpret in the context of other problem domains.

5.2 Geometric interpretation

Since the rows of H have no natural interpretation as axes, there is no natural interpretation of NMF geometrically. Nevertheless, it can be useful to plot the entries of either matrix as if they were coordinates. Two points that are located far apart must be dissimilar, but two points located close together are not necessarily similar. Because of the additive nature of the model, distance from the origin is important, because the only way for a point to be far from the origin is either to use parts with large magnitude entries, or to use large mixing coefficients or both. Conversely, a point can be close to the origin only if both the entries of its parts are small, and its mixing coefficients are small.

5.3 Graph interpretation

The graph interpretation of an NMF is a tripartite graph, with one set of nodes corresponding to objects, a second set corresponding to the r components, and a third set corresponding to the attributes. The differences in this case are that the graph is sparse because the two decomposition matrices are sparse; and that the constraints on the weights of edges are all based on sums of non-negative quantities, so they can be bounded more easily. Graphs whose edges have only positive weights are also inherently easier to understand.

5.4 Applying an NNMF

5.4.1 Selecting factors

NMF does not order components in any particular order, so it is not trivial to select 'interesting' or 'important' factors. When the rows of H are normalized so that the row sums are unity, the norms of the columns of W can be used as one way to order the components. These norms represent the extent to which each particular factor plays a role in the description of all of the objects. While this is a sensible measure, it is not clear that it captures the real property of interest, since we expect most parts to play a role in representing only a few objects. Another possibility is to use the Frobenius norm of each component, that is the product of the ith column of W and the ith row of H. This represents the mass of the ith bicluster, which should be meaningful in most contexts.

5.4.2 Denoising

Little attention has been paid to the effects of noise in NMF, partly because many applications start from integral, non-negative data, where noise is easy to see and remove in the raw data. If noise is widespread in the data, that is most values in the dataset have been slightly altered as the result of noise, then it is not clear what happens in the decomposition. Especially for those algorithms that enforce sparsity, it seems unlikely that noise will appear in one or a few components. Instead, it may be spread throughout the other, meaningful components.

Simple experiments suggest that NMF is sensitive to noise, especially to small but widely distributed noise, so that the biclustering structure changes substantially when only modest Gaussian distributed noise is introduced. This issue needs further research.

5.4.3 Similarity and clustering

The rows of W can be used as the basis of clustering the objects. This would be expected to perform better than clustering directly on the rows of V both because the rows of W are of length k rather than n, and because the sparsification of both W and H should make it easier to find meaningful boundaries within the geometric space containing the rows of W. Similarity is now based on the similarity of mixture coefficients rather than similarity of properties of the objects. Of course, this assumes implicitly that the parts have been properly discovered.

5.5 Algorithm issues

Because of the relatively large number of algorithms proposed to compute the NMF, little can be said in general about complexity. Lin [81] makes the point that algorithms seem to exist on a continuum, one end of which contains algorithms for which the cost of each step of the minimization is high but which require few steps; and the other end of which contains algorithms for which the step cost is small, but which require many steps. The constants may be large, and memory hierarchy effects may become significant for large matrices since the access pattern does not have spatial locality.

The NMF method proposed by Lee and Seung [67] is based on multiplicative update rules of W and H. This scheme is referred to as the multiplicative method (MM).

MM Algorithm

(1) Initialize W and H with nonnegative values.
(2) Iterate for each c, j and i until convergence or after l iterations:

(a) $H_{cj} \leftarrow H_{cj} \frac{(W^T V)_{cj}}{(W_T W H)_{cj} + \varepsilon}$

(b) $W_{ic} \leftarrow W_{ic} \frac{(V H^T)_{ic}}{(W H H_T)_{ic} + \varepsilon}$

In steps 2 (a) and (b), ε small positive parameter, is added to avoid division by zero. As observed from the MM Algorithm, W and H remain nonnegative during the updates. Simultaneous updating of W and H generally yield better results than updating each matrix factor fully before the other. In the algorithm, the columns of W or the basis vectors are normalized at each iteration; in case of W, the optimization is performed on a unit hypersphere with the columns of W effectively being mapped to the surface of the hypersphere by repeated normalization [68].

6 Using Matrix decomposition in Concept Lattice

In this part I will use matrix decompositions for improve concept lattice reduction. The goal is to minimize input data before constructing the concept lattice. The reduced input data has lower dimension than the original data and computing the lattice leads in creating smaller graphic representation in less time.

My idea is to use matrix decompositions as improved factor to concept lattice reduction. I will prove that the conflation between SVD or NMF with Concept lattice, will decrease the reduction of concept lattice.

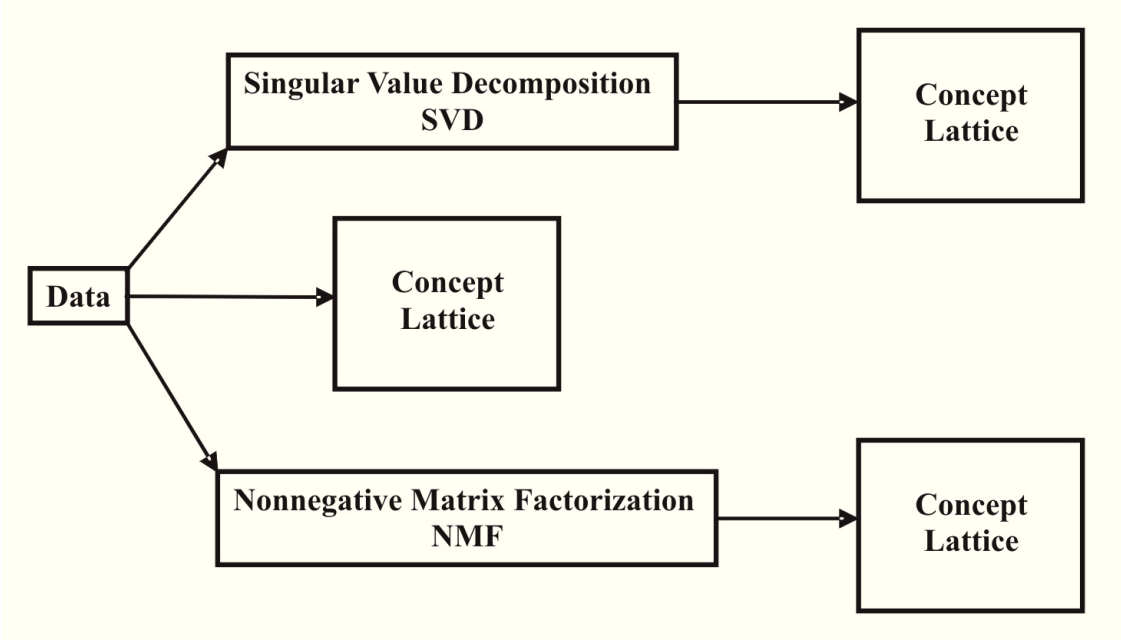


Figure 6.1: Using SVD and NMF as improved factor to Concept Lattice reduction

In this experiment I can show the behavior difference of the concept lattice reduction after using various methods of matrix decompositions. In this case I am using two methods for matrix decomposition, singular value decomposition (SVD) and Non-negative Matrix Factorization (NMF).

This experiment contain a several steps:

6.1 Computing concept lattices from the original data

In this step of experiment used a sample random data. This data do not conform to reality and was created randomly for this experiment. The best way to represent input data is to write them like a matrix A with 20 rows and 6 columns. It is not hard to imagine them like adjacency matrix of 20 documents and 6 terms.

Each $A_{i,j}$ denotes that the term i is present in document j.

	A0	A1	A2	A3	A4	A5
O0	0	0	0	0	0	1
O1	1	0	0	1	0	1
O2	0	0	0	1	0	1
O3	0	0	0	1	0	0
O4	1	0	0	1	0	0
O5	1	0	0	0	0	0
O6	0	1	1	1	1	0
O7	0	0	0	1	1	0
O8	0	1	1	0	0	0
O9	0	1	0	1	1	0
O10	0	0	1	1	1	0
O11	0	1	1	0	1	0
O12	0	1	1	1	0	0
O13	0	1	0	1	0	0
O14	0	1	0	0	1	0
O15	0	0	1	1	0	0
O16	0	0	1	0	1	0
O17	1	0	0	1	1	0
O18	0	1	1	0	0	1
O19	1	1	1	0	0	1

Figure 6.2: Formal context from the original data

Figure 6.2 shows the context input data for concept lattice, where the rows show the objects and the columns show the attributes for these objects. The values 1 and 0 tell us if this attribute is exist in this object or not.

(o0, o1, o2, o3, o4, o5, o6, o7, o8, o9, o10, o11, o12, o13, o14, o15, o16, o17, o18, o19) () 25 node
(o1, o2, o3, o4, o6, o7, o9, o10, o12, o13, o15, o17) (a3) 2 node
(o6, o8, o10, o11, o12, o15, o16, o18, o19) (a2) 5 node
(o6, o8, o9, o11, o12, o13, o14, o18, o19) (a1) 9 node
(o6, o7, o9, o10, o11, o14, o16, o17) (a4) 1 node
(o6, o8, o11, o12, o18, o19) (a1, a2) 13 node
(o6, o7, o9, o10, o17) (a3, a4) 4 node
(o1, o4, o5, o17, o19) (a0) 18 node
(o0, o1, o2, o18, o19) (a5) 0 node
(o6, o10, o12, o15) (a2, a3) 7 node
(o6, o9, o11, o14) (a1, a4) 10 node
(o6, o9, o12, o13) (a1, a3) 11 node
(o6, o10, o11, o16) (a2, a4) 6 node
(o1, o4, o17) (a0, a3) 20 node
(o6, o11) (a1, a2, a4) 15 node
(o6, o9) (a1, a3, a4) 12 node
(o18, o19) (a1, a2, a5) 14 node
(o1, o2) (a3, a5) 3 node
(o6, o10) (a2, a3, a4) 8 node
(o1, o19) (a0, a5) 19 node
(o6, o12) (a1, a2, a3) 16 node
(o17) (a0, a3, a4) 22 node
(o19) (a0, a1, a2, a5) 23 node
(o1) (a0, a3, a5) 21 node
(o6) (a1, a2, a3, a4) 17 node
() (a0, a1, a2, a3, a4, a5) 24 node

Figure 6.3: Nodes content in Concept Lattice

All the matrixes passed as input for Concept Lattice software which was developed at VSB-TU Ostrava. The Concept Lattice software produced a concept lattice. The view of these concept lattices was completed with GraphPlace software. The output of GraphPlace is a Postscript file that can be viewed in any postscript viewer.

Figure 6.4 shows the concept lattice computed from the context for objects and attributes in figure 6.2. We can see the difficulty for reading this concept lattice. In this example I use only 20×6 matrix like input data, we can imagine the difficulty and the complexity if the input data are bigger.

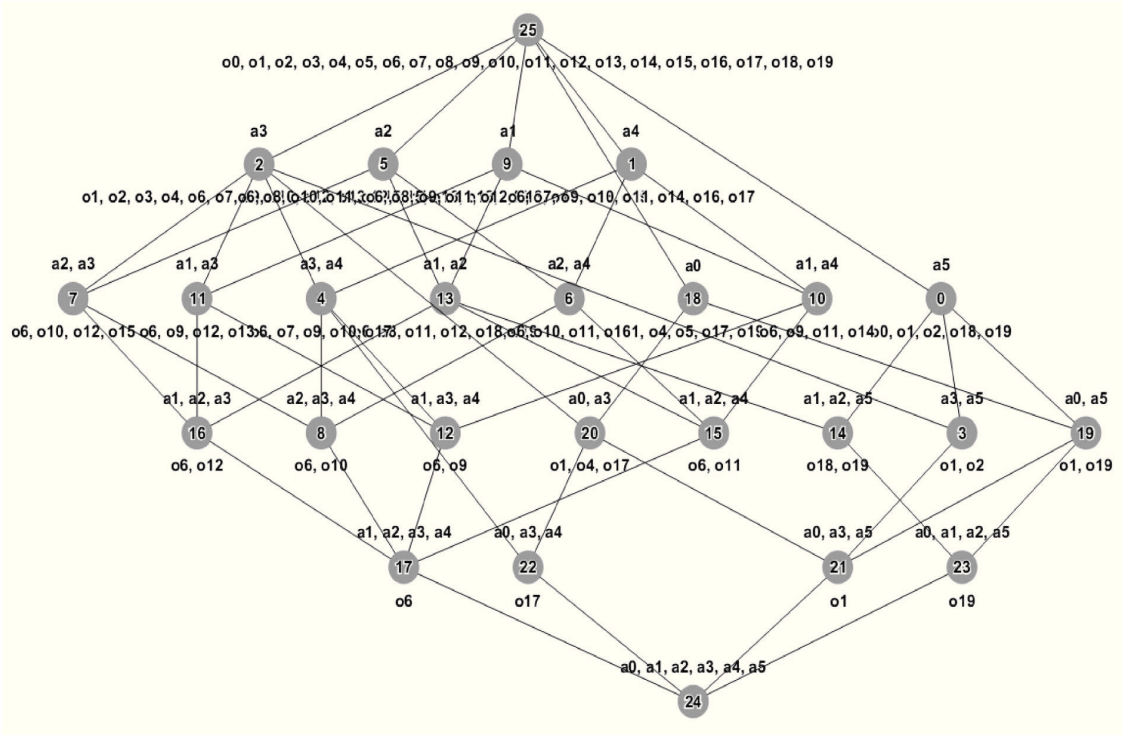


Figure 6.4: Concept lattice computed from simplified formal context (Figure 6.2)

6.2 Computing SVD

SVD computations were made by SVDLIBC-fix software. This software is free and is written in C language. An incidence matrix acts as the input. Software can compute all three matrixes: U, S and V^T. k - rank was chosen $k = 4$.

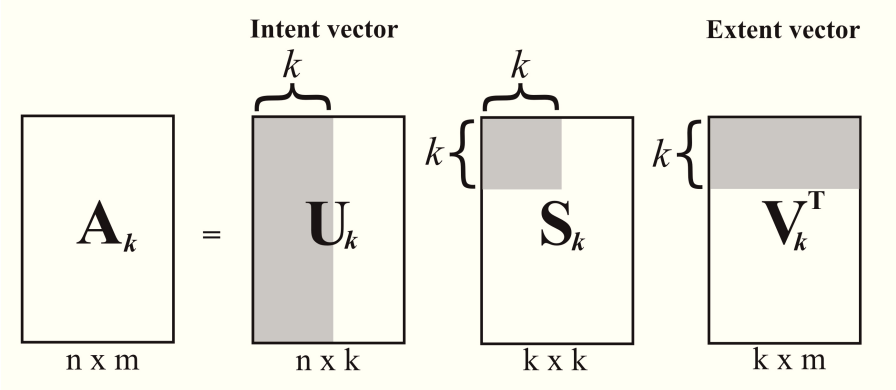
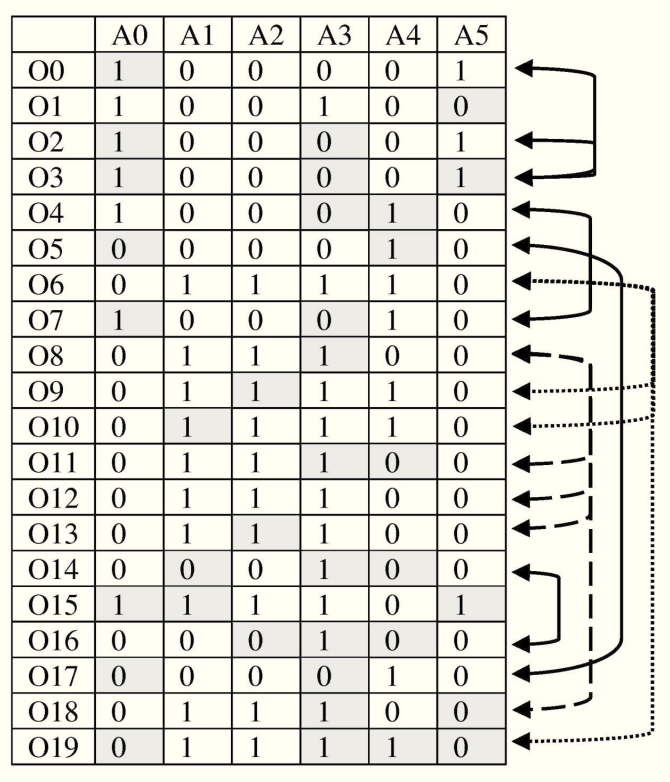


Figure 6.5: k-rank for singular value decomposition

After using SVD with good choice k-rank, we can see that the output data from the SVD method are changed and brewed new attributes combinations in the objects, these combinations repeated in more than one object (figure 6.6). When discovering output after using SVD, we have found changes in the objects content (SVD method adds and removes some attributes).



	A0	A1	A2	A3	A4	A5
O0	1	0	0	0	0	1
O1	1	0	0	1	0	0
O2	1	0	0	0	0	1
O3	1	0	0	0	0	1
O4	1	0	0	0	1	0
O5	0	0	0	0	1	0
O6	0	1	1	1	1	0
O7	1	0	0	0	1	0
O8	0	1	1	1	0	0
O9	0	1	1	1	1	0
O10	0	1	1	1	1	0
O11	0	1	1	1	0	0
O12	0	1	1	1	0	0
O13	0	1	1	1	0	0
O14	0	0	0	1	0	0
O15	1	1	1	1	0	1
O16	0	0	0	1	0	0
O17	0	0	0	0	1	0
O18	0	1	1	1	0	0
O19	0	1	1	1	1	0

Figure 6.6: Simplified formal context after using SVD with k-rank

Some attributes are added and removed in some objects to be similar with another objects. We see that attribute $A0$ is added to the objects $(O1, O2, O3, O7, O15)$ and removed from the objects $(O5, o17, O19)$, attributes $A1$ is added to the objects $(O10, O15)$ and removed from object $(O14)$, attribute $A2$ is added to the objects $(O9, O13)$ and removed from the object $(O16)$, attribute $A3$ is added to the objects $(O8, O11, O14, O16, O18, O19)$ and removed from the objects $(O2, O3, O4, O7, O17)$, attribute $A4$ is added to the objects $(O4, O5, O19)$ and removed from the objects $(O11, O14, O16)$ and attributes $A5$ is added to the objects $(O3, O15)$ and removed from the objects $(O1, O18, O19)$.

Adding and deleting these attributes in objects brewed new attributes combinations in the objects, and these combination are repeated in more than one objects. We can see that the objects $(O0, O2, O3)$ have the same attributes $(A0, A5)$, objects $(O4, O7)$ have the same attributes $(A0, A4)$, objects $(O5, O17)$ have the same attribute $(A4)$, objects $(O6, O9, O10, O19)$ have the same attributes $(A1, A2, A3, A4)$, objects $(O8, O11, O12, O13, O18)$ have the same attributes $(A1, A2, A3)$, objects

($O14, O16$) have the same attribute ($A3$).

From this repeating of attributes combination in the objects, which is not needed to represent all the objects with the same attributes, we can only represent the objects that have peculiar combinations of attributes (figure 6.7).

This process minimizes the representing data to 40% from the original data with saving the primary attributes in the objects.

	A0	A1	A2	A3	A4	A5
A	1	0	0	1	0	0
B	1	0	0	0	0	1
C	1	0	0	0	1	0
D	0	0	0	0	1	0
E	0	1	1	1	1	0
F	0	1	1	1	0	0
G	0	0	0	1	0	0
H	1	1	1	1	0	1

Figure 6.7: Representing data after combining the similar Objects

Where the A, B, C, D, E, F, G and H are groups of the same objects after using SVD on the original data:

Groups	Objects	Attributes
A	O1	A0, A3
B	O0, O2, O3	A0, A5
C	O4, O7	A0, A4
D	O5, O17	A4
E	O6, O9, O10, O19	A1, A2, A3, A4
F	O8, O11, O12, O13, O18	A1, A2, A3
G	O14, O16	A3
H	O15	A0, A1, A2, A3, A5

Figure 6.8: Groups content

6.3 Computing concept lattices from data after using SVD

Minimizing the representing data of the original data in the simplified formal context after using SVD with k-rank, give us minimizing of view presenting the concept lattice which present those data (see figure 6.9).

After using the SVD, the concept lattice is reduced and the number of nodes in the concept lattice after using SVD is less then the number of nodes in the concept lattice before using SVD. The original concept lattice has 26 nodes, while the concept lattice after using SVD has only 11 nodes. This decrease of nodes come from the combining of similar objects in the simplified formal context after using SVD.

Figure 6.10. shows the nodes content in concept lattice after using Singular Value decomposition (SVD).

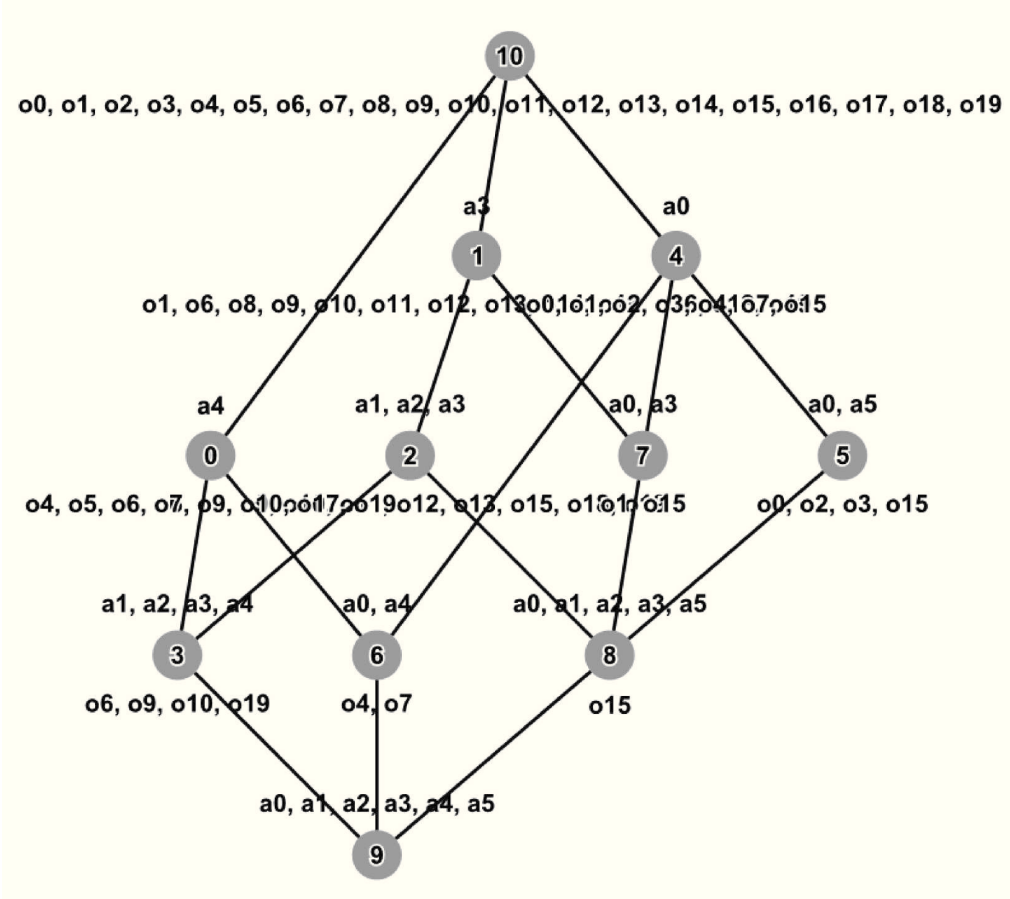


Figure 6.9: Concept lattice computed from simplified formal context after using SVD with k-rank (figure 6.6)

(o0, o1, o2, o3, o4, o5, o6, o7, o8, o9, o10, o11, o12, o13, o14, o15, o16, o17, o18, o19) () 10 node
(o1, o6, o8, o9, o10, o11, o12, o13, o14, o15, o16, o18, o19) (a3) 1 node
(o6, o8, o9, o10, o11, o12, o13, o15, o18, o19) (a1, a2, a3) 2 node
(o4, o5, o6, o7, o9, o10, o17, o19) (a4) 0 node
(o0, o1, o2, o3, o4, o7, o15) (a0) 4 node
(o0, o2, o3, o15) (a0, a5) 5 node
(o6, o9, o10, o19) (a1, a2, a3, a4) 3 node
(o1, o15) (a0, a3) 7 node
(o4, o7) (a0, a4) 6 node
(o15) (a0, a1, a2, a3, a5) 8 node
() (a0, a1, a2, a3, a4, a5) 9 node

Figure 6.10: Nodes content in concept lattice after using Singular Value decomposition (SVD)

The upper node (node 25) in the original concept lattice has 6 sub-nodes (nodes 0, 1, 2, 5, 9, 18), but in concept lattice after using SVD the upper node (node 10) has only 3 sub-nodes (nods 0, 1, 4),(see figures 6.11 and 6.12).

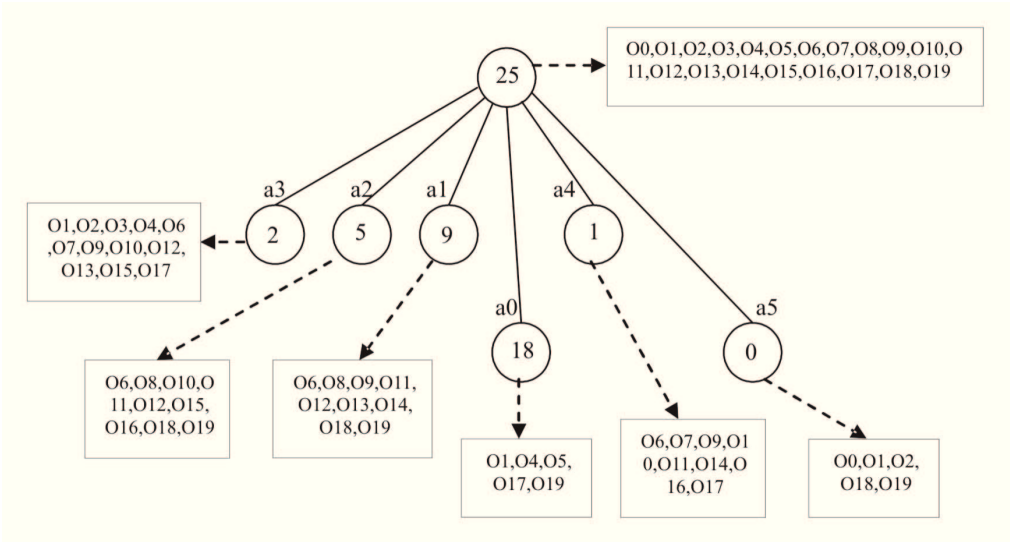


Figure 6.11: The upper node in the original concept lattice before using SVD

The reason for the decrease in the number of nodes after using SVD due to growth new combination of objects contains the same attributes.

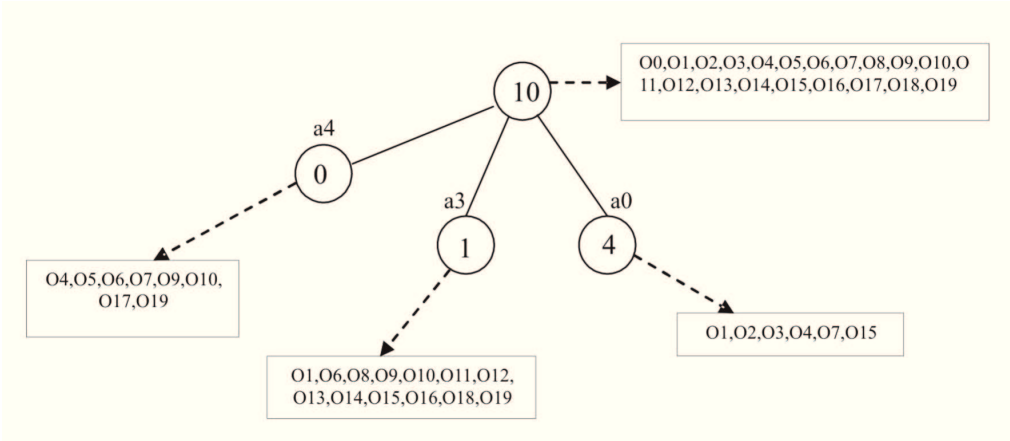


Figure 6.12: The upper node in the concept lattice after using SVD

The attribute A3 added to the objects $(O8, O11, O14, O16, O18, O19)$ and deleted from the objects $(O2, O3, O4, O7, O17)$, this change gives us that the node 2 in the original lattice has a new combination of objects in the lattice after using SVD $(O1, O6, O8, O9, O10, O11, O12, O13, O14, O15, O16, O18, O19)$.

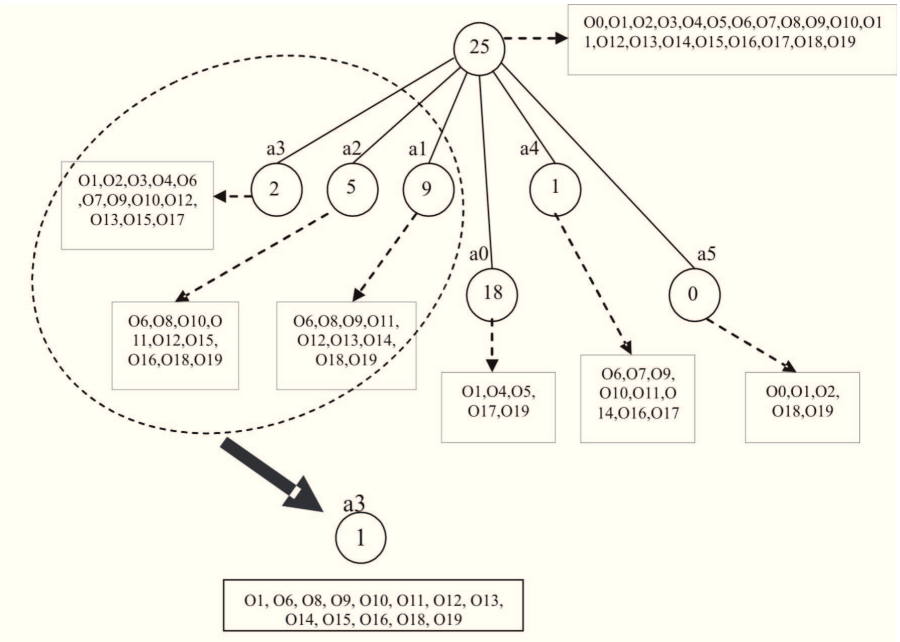


Figure 6.13: Nodes 2, 5 and 9 in the original concept lattice come under nod 1 in the new lattice after using SVD

The combination of objects in the node 5 in the original lattice has changed to $(O6, O8, O9, O10, O11, O12, O13, O15, O18, O19)$ and the combination of objects in the node 9 to $(O6, O8, O9, O10, O11, O12, O13, O15, O18, O19)$. Therefore, the nodes 2, 5 and 9 in the original lattice come under node 1 in the lattice after using

SVD, that's content the combination of these nodes (see figur 6.13).

After using SVD, attribute A0 is added to the objects $(O0, O2, O3, O7, O15)$ and deleted from the objects $(O5, O17, O19)$, this change gives us that the node 18 in the original lattice (concept lattice before using SVD) has a new combination of objects in the lattice after using SVD $(O0, O1, O2, O3, O4, O7, O15)$. And the combination of objects in the node 0 in the original lattice has changed to $(O0, O2, O3, O15)$. For that the nodes 0 and 18 in the original lattice come under node 4 in the lattice after using SVD, that's content the combination of these nodes(see figure 6.14).

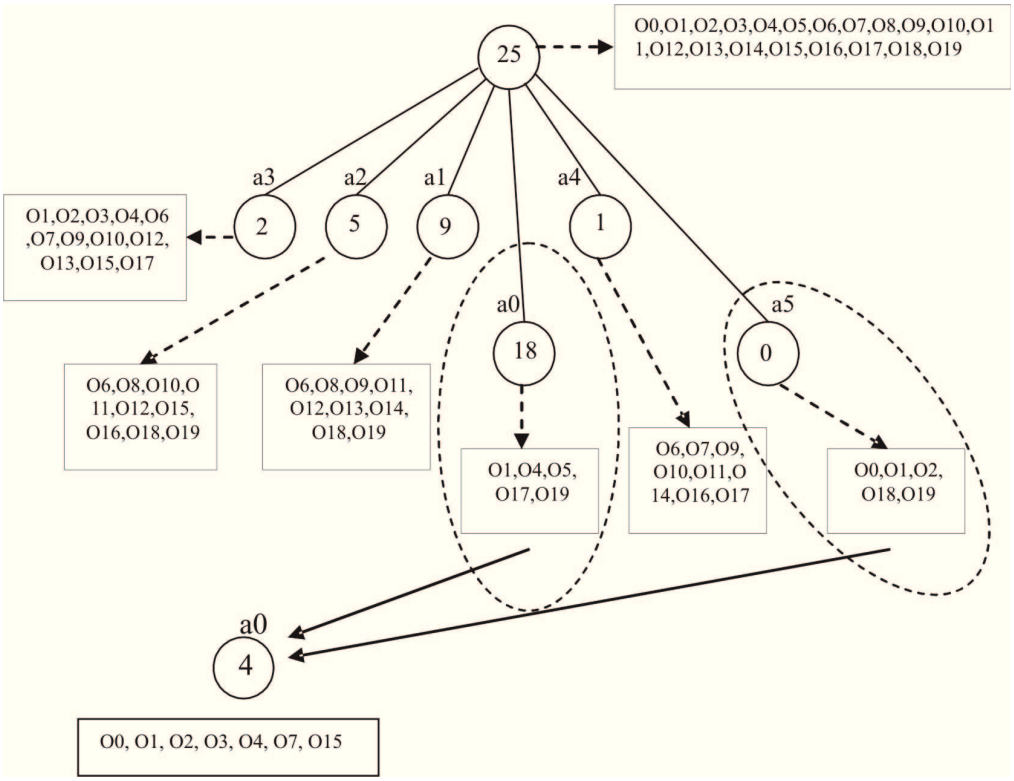


Figure 6.14: Nodes 0 and 18 in the original concept lattice come under node 4 in the new lattice after using SVD

We can see that the node 3 from the original concept lattice was deleted, because the attributes composition ($A3$ and $A5$) in the objects $(O1, O2)$ is not available because after using SVD the attribute A5 was deleted from the object $O1$ and the attribute $A3$ was deleted from the object $O2$.

We also can see that after using SVD, some attributes are removed and added, and more objects have the same compositions of attributes. The node 11 has a composition of attributes ($A1$ and $A3$) in the objects $(O6, O9, O12, O13)$, these composition of attributes $(A1, A3)$ existed in the objects $(O8, O10, O11, O18, O19)$ too. For that the nodes 4, 7 and 11 in the original lattice come under the node 2 in the new lattice after using SVD (see figure 6.15).

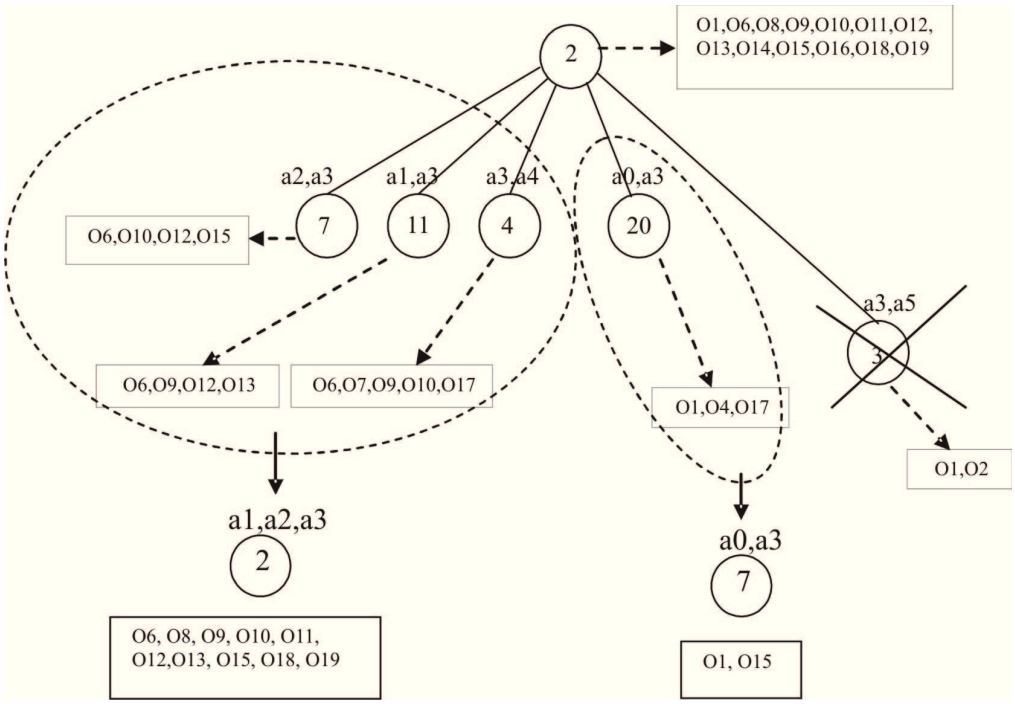


Figure 6.15: Node 3 is deleted. Nodes 4, 7 and 11 in the original concept lattice come under node 4 in the new lattice after using SVD

6.4 Computing NMF

NMF computations were done by a software developed on VSB-TU Ostrava for this concept lattices research. The software is using multiplicative method proposed by Lee and Seung. The goal is to find a low rank approximation of V in terms of some matrix by factoring V into product of two reduced-dimensional matrixes W and H. Dimensions of W and H are $n \times k$ and $k \times m$, where k is reduced rank.

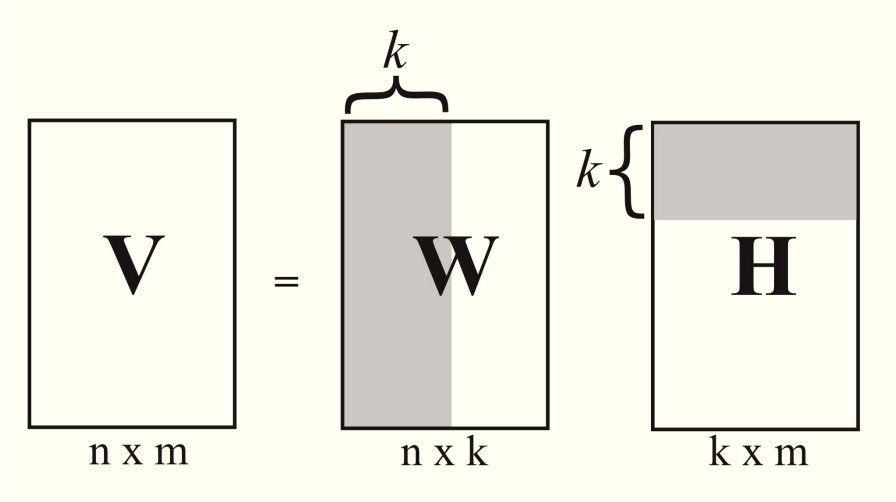


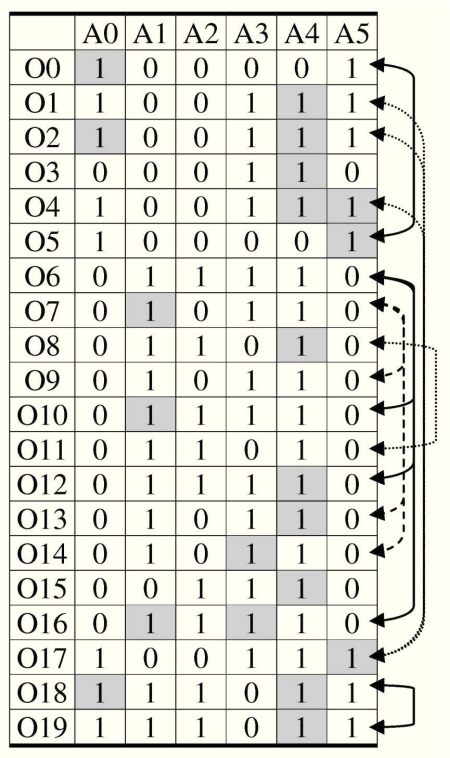
Figure 6.16: k-reduced Nonnegative Matrix Factorization (NMF)

After using NMF with good choice k-rank, we can see that the output data from the NMF method are changed and brewed new attributes combinations in the objects, these combinations repeated in more than one object (figure 6.17). When discovering the output after using NMF, we have found that objects has in average more attributes (NMF method adds some attributes to the objects).

Some attributes are added to some objects to be similar with another objects, we can see that attribute $A0$ is added in the objects $(O0, O2, O18)$, attribute $A1$ is added in the objects $(O7, O10, O16)$, attribute $A3$ is added in the objects $(O14, O16)$. we can see also that the attribute $A4$ is added in the objects $(O1, O2, O3, O4, O8, O12, O13, O15, O18, O19)$ and attribute $A5$ is added in the objects $(O4, O5, O17)$.

Added to these attributes in objects produced new attributes combinations in the objects. These combinations repeated in more than one object. We can see that the objects $(O0, O5)$ have the same attributes $(A0, A5)$, objects $(O1, O2, O4, O17)$ have the same attributes $(A0, A6, A7, A9)$, objects $(O6, O10, O12, O16)$ have the same attributes $(A1A2, A3, A4)$, objects $(O7, O9, O13, O14)$ have the same attributes $(A1, A3, A4)$, objects $(O8, O11)$ have the same attribute $(A1, A2, A4)$, objects $(O18, O19)$ have the same attributes $(A0, A1, A2, A4, A5)$.

From this repetition of attributes combination in the objects, not which not



	A0	A1	A2	A3	A4	A5
O0	1	0	0	0	0	1
O1	1	0	0	1	1	1
O2	1	0	0	1	1	1
O3	0	0	0	1	1	0
O4	1	0	0	1	1	1
O5	1	0	0	0	0	1
O6	0	1	1	1	1	0
O7	0	1	0	1	1	0
O8	0	1	1	0	1	0
O9	0	1	0	1	1	0
O10	0	1	1	1	1	0
O11	0	1	1	0	1	0
O12	0	1	1	1	1	0
O13	0	1	0	1	1	0
O14	0	1	0	1	1	0
O15	0	0	1	1	1	0
O16	0	1	1	1	1	0
O17	1	0	0	1	1	1
O18	1	1	1	0	1	1
O19	1	1	1	0	1	1

Figure 6.17: Simplified formal context after using NMF with k-rank

required to represent all the objects with the same attributes, we can only represent the objects that have peculiar combinations of attributes (see figure 6.18).

This process minimizes the representing data to 40% from the original data with saving the primary attributes in the objects.

	A0	A1	A2	A3	A4	A5
A	1	0	0	0	0	1
B	1	0	0	1	1	1
C	0	0	0	1	1	0
D	0	1	1	1	1	0
E	0	1	0	1	1	0
F	0	1	1	0	1	0
G	0	0	1	1	1	0
H	1	1	0	1	1	1

Figure 6.18: representing data after combining the similar objects

Where the A, B, C, D, E, F, G and H are a groups of some objects after using NMF on the original data:

Groups	Objects	Attributes
A	O0, O5	A0, A5
B	O1, O2, O4, O17	A0, A3, A4, A5
C	O6,O10, O12, O16	A1, A2, A3, A4
D	O7, O9, O13, O14	A1, A3, A4
E	O8, O11	A1, A2, A4
F	O18, O19	A0, A1, A2, A4, A5
G	O3	A3, A4
H	O15	A2, A3, A4

Figure 6.19: Groups of content

6.5 Computing concept lattices from data after using NMF

Minimizing the representing data of the original data in the simplified formal context after using NMF with k-rank, give us minimizing of view presenting the concept lattice which present those data (see figure 6.20).

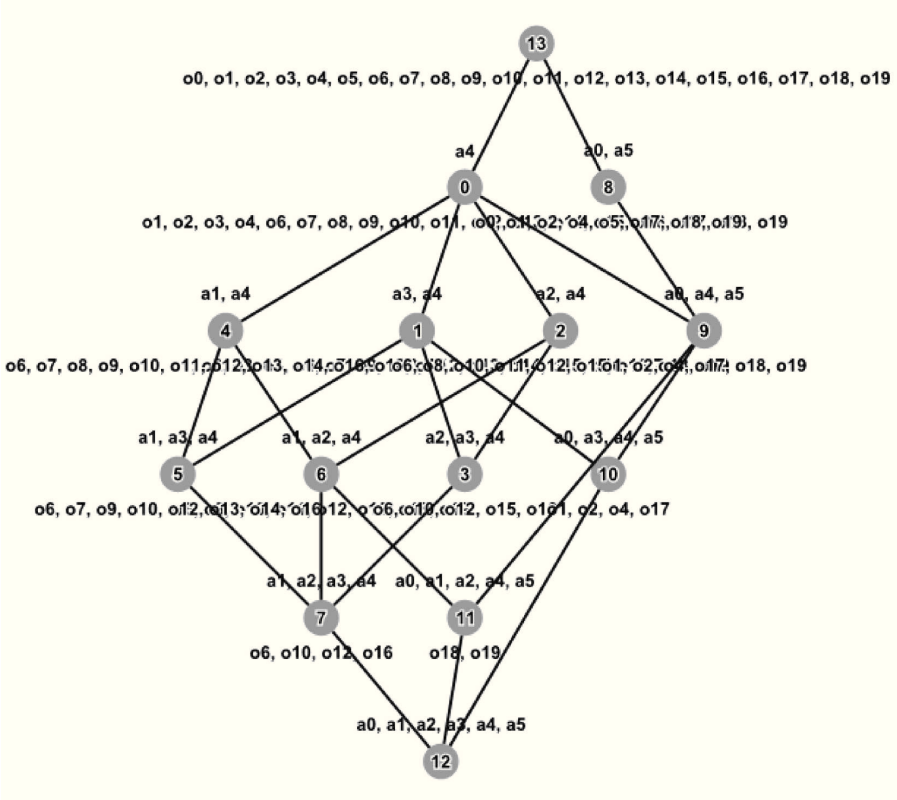


Figure 6.20: Concept lattice computed from simplified formal context after using NMF with k-rank (figure 6.17)

After using NMF the concept lattice is reduced, and the number of nodes in the concept lattice after using NMF is less then the number of nodes in the

concept lattice before using NMF. The original concept lattice has 26 nodes, while the concept lattice after using NMF has only 14 nodes. This decrease of node come from the combining of similar objects in the Simplified formal context after using NMF.

In figure 6.21, we see the nodes content in Concept Lattice after using Nonnegative Matrix Factorization (NMF).

(o0, o1, o2, o3, o4, o5, o6, o7, o8, o9, o10, o11, o12, o13, o14, o15, o16, o17, o18, o19) () 13 node
(o1, o2, o3, o4, o6, o7, o8, o9, o10, o11, o12, o13, o14, o15, o16, o17, o18, o19) (a4) 0 node
(o1, o2, o3, o4, o6, o7, o9, o10, o12, o13, o14, o15, o16, o17) (a3, a4) 1 node
(o6, o7, o8, o9, o10, o11, o12, o13, o14, o16, o18, o19) (a1, a4) 4 node
(o6, o8, o10, o11, o12, o15, o16, o18, o19) (a2, a4) 2 node
(o6, o8, o10, o11, o12, o16, o18, o19) (a1, a2, a4) 6 node
(o6, o7, o9, o10, o12, o13, o14, o16) (a1, a3, a4) 5 node
(o0, o1, o2, o4, o5, o17, o18, o19) (a0, a5) 8 node
(o1, o2, o4, o17, o18, o19) (a0, a4, a5) 9 node
(o6, o10, o12, o15, o16) (a2, a3, a4) 3 node
(o6, o10, o12, o16) (a1, a2, a3, a4) 7 node
(o1, o2, o4, o17) (a0, a3, a4, a5) 10 node
(o18, o19) (a0, a1, a2, a4, a5) 11 node
() (a0, a1, a2, a3, a4, a5) 12 node

Figure 6.21: Nodes content in Concept Lattice after using Nonnegative Matrix Factorization (NMF)

The upper node (node 25) in the original concept lattice has 6 sub-nodes (nodes 0, 1, 2, 5, 9, 18), but in concept lattice after using NMF the upper node (node 13) has only 2 sub-nodes (nods 0, 8), (see figures 6.22 and 6.23).

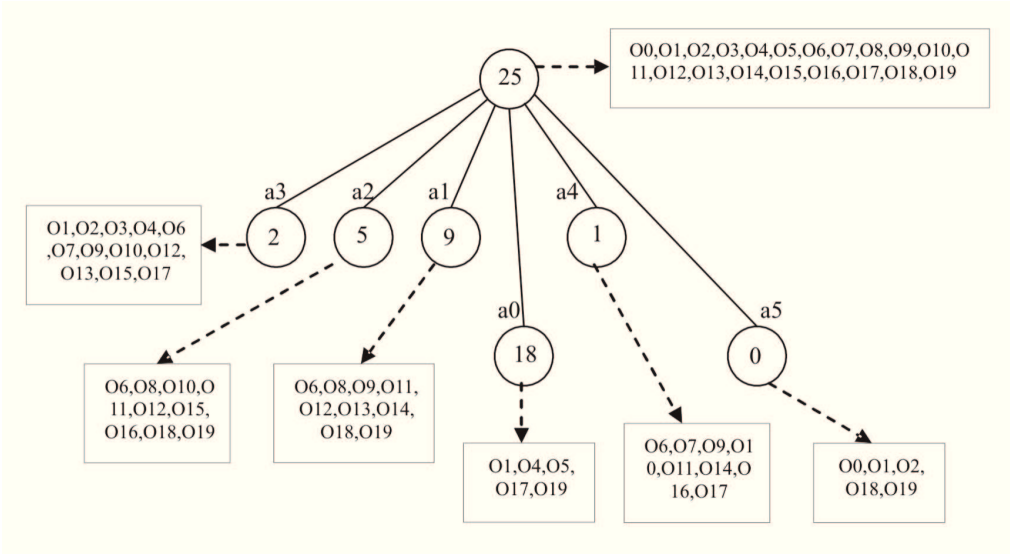


Figure 6.22: The upper node in the original concept lattice before using NMF

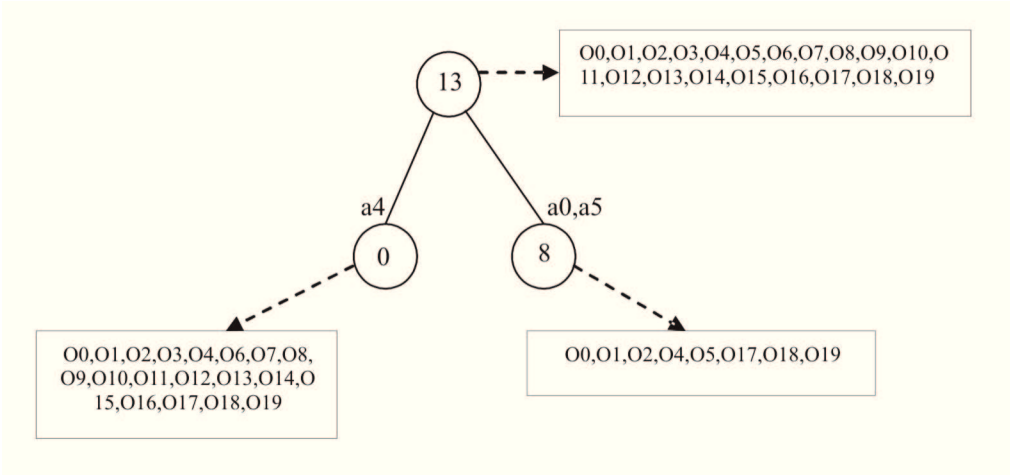


Figure 6.23: The upper node in the concept lattice after using NMF

The reason for the decrease in the number of nodes after using NMF is the growth the new combinations of objects contains the same attributes.

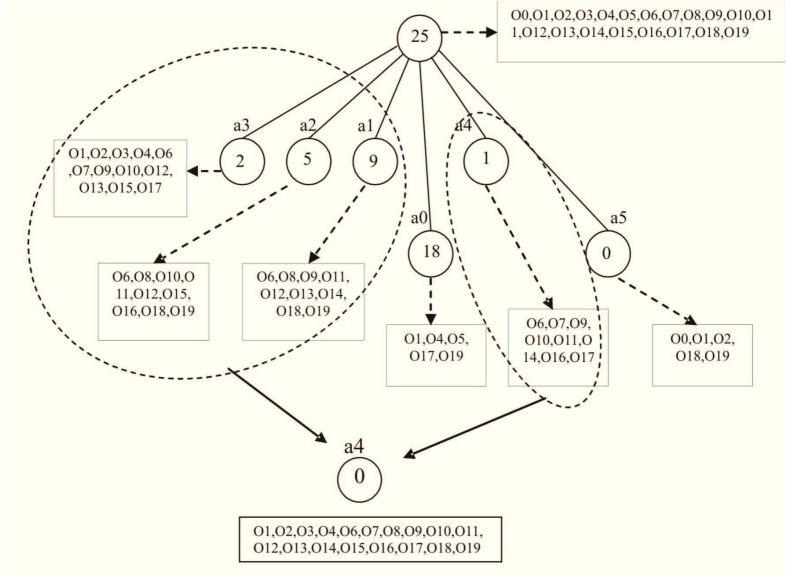


Figure 6.24: Nodes 2, 5, 9 and 1 in the original concept lattice came under node 0 in the new lattice after using NMF

The attribute A4 added to all of these objects in the data after using NMF $(O1, O2, O3, O4, O8, O12, O13, O15, O18, O19)$. And the node 0 in the new lattice (concept lattice after using NMF) contains all the objects that contains the attribute $A4$ in $(O1, O2, O3, O4, O6, O7, O8, O9, O10, O11, O12, O13, O14, O15, O16)$ and objects $(O17, O18, O19)$. For that the nodes 2, 5 and 9 in the original concept lattice came under node 0 in the new lattice (concept lattice after using NMF) that contains the components of these nodes (figure 6.24).

After using NMF on the original lattice, the attribute $A5$ is added to the objects $(O4, O5, O17)$ and the attribute A0 is added to the objects $(O0, O2, O18)$. After this addition, the node 0 with attribute $A5$ in the original concept lattice contains the objects $(O0, O1, O2, O4, O5, O17, O18, O19)$.

The node 18 with attribute $A0$ in the original lattice contains the objects $(O0, O1, O2, O4, O5, O17, O18, O19)$. The two nodes contains the same object, for that the two nodes come under the node 8 in the new lattice (concept lattice after using NMF) that contains the obecjts with attributes $A0$ and $A5$ (figure 6.25).

In another case of decreasing in the number of nodes after using NMF, that the nodes 3, 14, 19 and 20 in the original concept lattice come under node 9 in the new lattice (concept lattice after using NMF), (see figure 6.27).

In the original concept lattice node 3 with attributes $(A3, A5)$ in objects $(O1, O2)$, the node 14 with attributes $(A1, A2, A5)$ in the objects $(O18, O19)$, node 19 with attributes $(A0, A5)$ in the objects $(O1, O19)$ and node 20 with attributes $(A0, A3)$ in the objects $(O1, O4, O17)$.

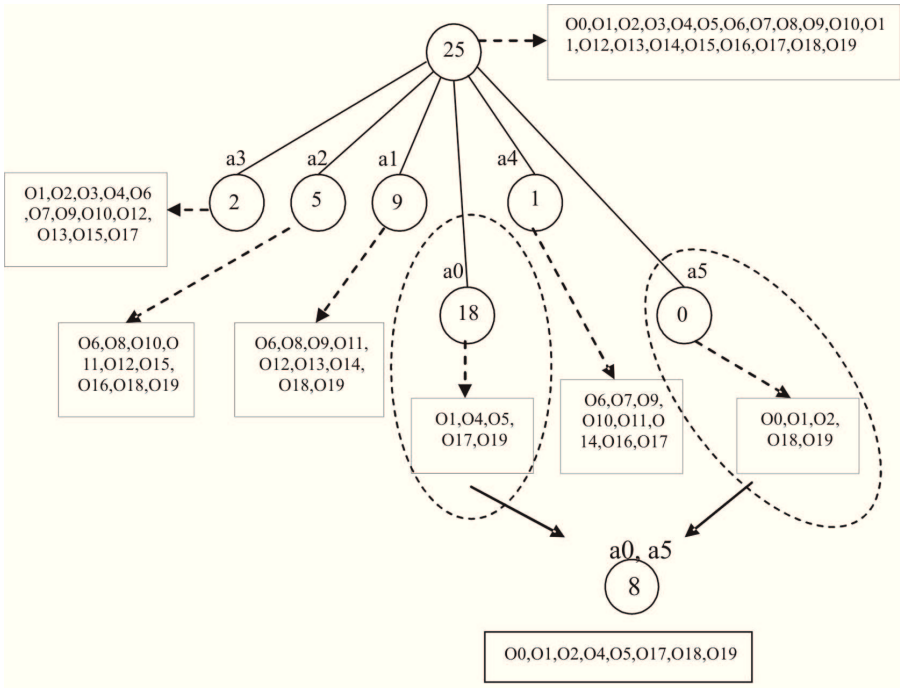


Figure 6.25: Nodes 0 and 18 in the original concept lattice came under node 8 in the new lattice after using NMF

After using NMF, the attribute A0 is added to the objects $(O2, O18)$, the attribute $A4$ is added to the objects $(O1, O2, O4, O18, O19)$ and the attribute $A5$ is added to the objects $(O4, O17)$.

After using NMF the objects that make up the nodes 3, 14, 19 and 20 in the original concept lattice, contains new combinations of attributes.

From the figure 6.26 we can see, that all the objects have the same combination of attributes $(A0, A4, A5)$, and the attribute $(A1, A2, A3)$ were lost when

Objects	Attributes
O1	A0, A4, A5
O2	A0, A3, A4, A5
O4	A0, A3, A4, A5
O17	A0, A3, A4, A5
O18	A0, A1, A2, A4, A5
O19	A0, A1, A2,A4, A5

Figure 6.26: combination of attributes

nodes 2, 5 and 9 in the original concept lattice came under node 0 in the new lattice (concept lattice after using NMF).

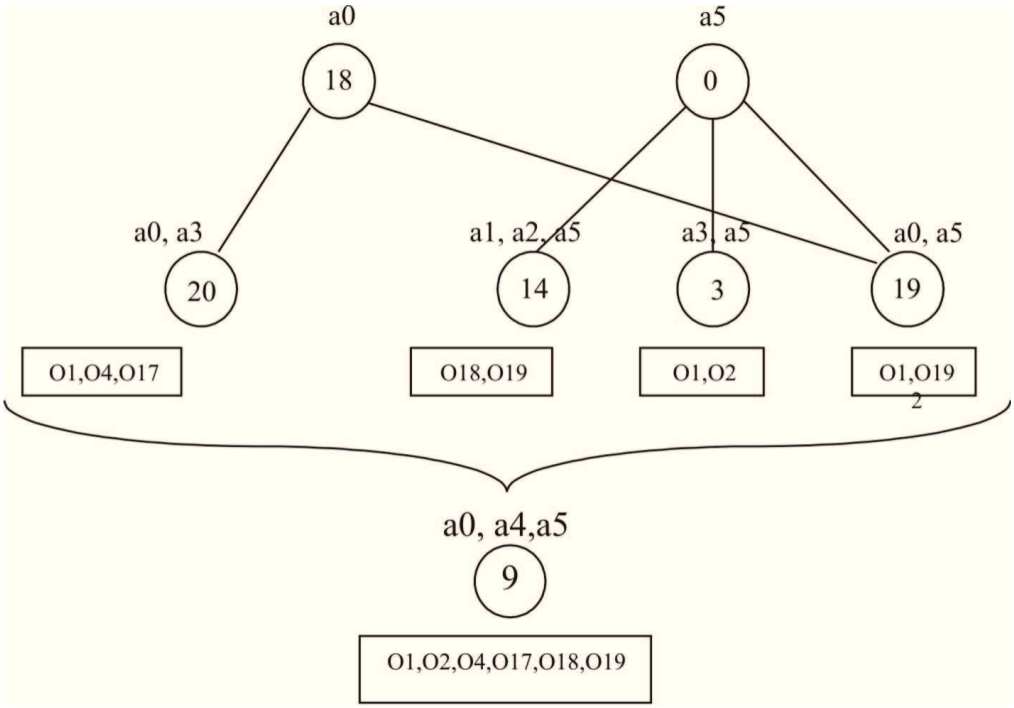


Figure 6.27: Nodes 3, 14, 19 and 20 in the original concept lattice came under node 9 in the new lattice

Every node in concept lattice has a distinct set of attributes. This means that there can not be found two nodes with the same set of attributes. The experiment consist in finding and describing changes of concepts after using NMF. It's found that NMF has added some attributes into concept's set of attributes. Thanks to this adding of attributes, we have different collections of concepts with same attributes.

6.6 Behavior of the Concept Lattice Reduction to visualizing data after Using Matrix Decompositions

After using SVD and NMF on the data, every method has different way to decompose the data.

The Singular Value Decomposition (SVD) depends on deleting and adding the secondary attributes from the objects in the original data. This way can give us minimum primary attributes to collect more objects has the same composition of attributes. On the other hand, the Nonnegative Matrix Factorization (NMF) depends on adding some secondary attributes to the objects in the original data to equalize the attributes between some of the objects so to collect more objects that have the same composition of attributes.

From the experiment, we can see the behavior of the Concept Lattice Reduction to visualizing data after Using Matrix Decompositions in the two methods. when using the SVD, the Concept Lattice delete the nodes, because the SVD delete and add attributes from the objects, and not existing relation between the deleted nodes with the another's nodes.

But when using the NMF, the concept lattice collects some nodes in one node, because the NMF add some attributes to some objects.

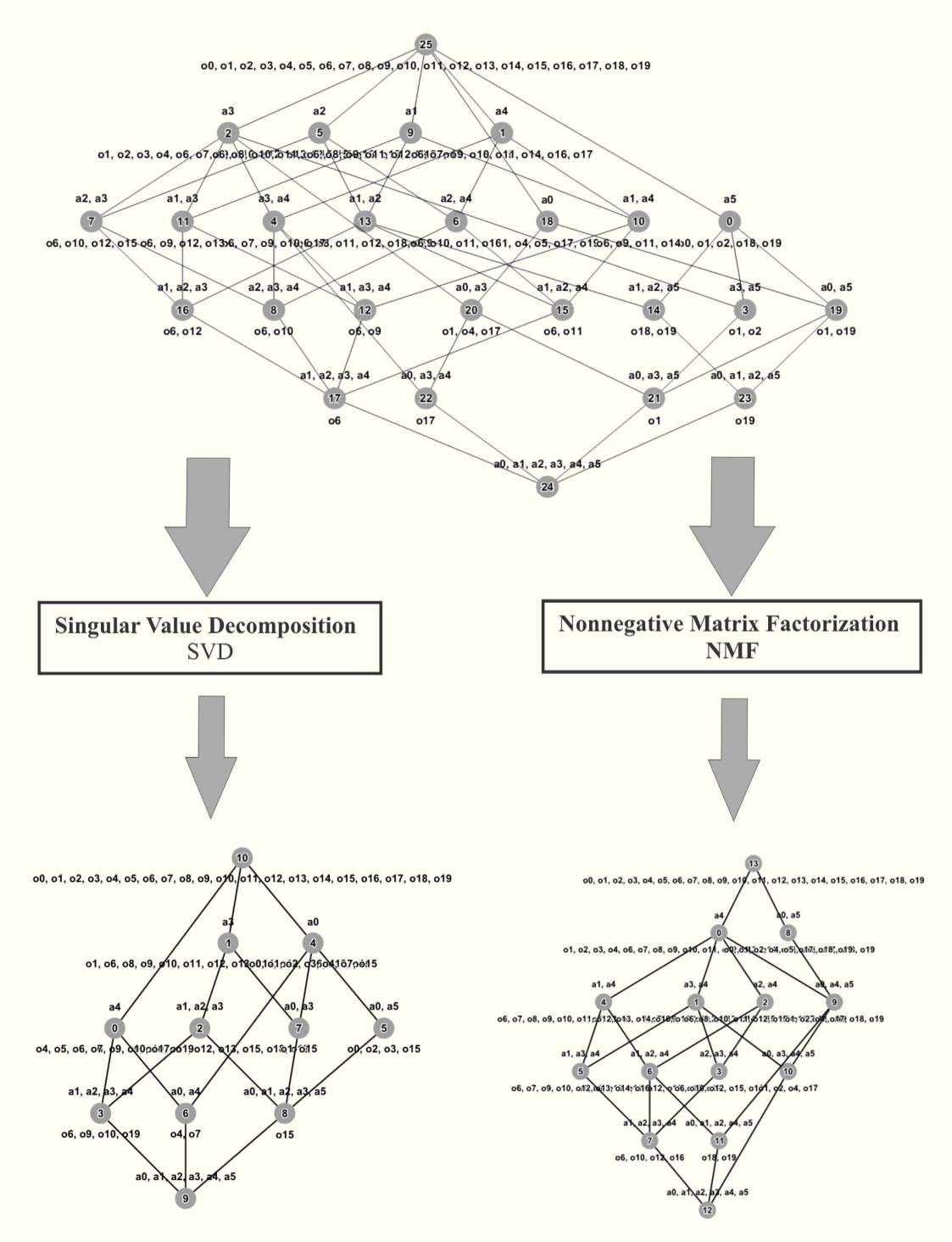


Figure 6.28: behavior of the Concept Lattice Reduction to visualizing data after Using Matrix Decompositions methods

7 Using Concept Lattice and Matrix decomposition in Search Results Clustering

In the last years, the search result clustering has attracted a substantial amount of research (e.g., information retrieval, machine learning, human-computer interaction, computational linguistics, data mining, formal concept analysis, graph drawing).

Search result clustering groups search results by topic. Thus provides us with complementary view to the information returned by big documents ranking systems. This approach is especially useful when document ranking fails to give us a precise result. This method allows a direct access to a subtopic; search result clustering reduces the information, helps filtering out irrelevant items, and favors exploration of unknown or dynamic domains. Search result clustering is different from the conventional document clustering. When clustering takes place as a post-processing step on the set of results retrieved by an information retrieval system on a query, it may be both more efficient, because the input consists of few hundred of snippets, and more effective, because query-specific text features are used. On the other hand, search result clustering must fulfill a number of more stringent requirements raised by the nature of the application in which it is embedded[69].

In this part, I will present the concept lattice with matrix decomposition (Singular Value Decomposition SVD) to be used in the search result clustering. I will use the Singular Value Decomposition as a mathematical method to reduce a big value of objects by combining the attributes of these objects.

7.1 Problem formalization and algorithm

The distinctive characteristic of the algorithm is that it identifies meaningful cluster labels and only then assigns search results to these labels to build proper clusters. The algorithm consists of five steps:

1. Pre-processing the input snippets, which includes tokenization, stemming and stop-word marking.
2. Identifies words and sequences of words frequently appearing in the input snippets.
3. Concept Lattice and Matrix decomposition is used to induce cluster labels.
4. Snippets are assigned to each of these labels to form proper clusters.
5. Post-processing, which includes cluster merging and pruning.

The step 3 is the core of the algorithm, because this step relies on the Vector Space Model and a term-document matrix A having n rows, where n is the number of input snippets, and m columns, where m is the distinct words found in the input snippets. Each element A_{nm} of A numerically represents the relationship between word m and snippet n.

The concept lattice and matrix decomposition may be applied on the binary matrix A which created from the step 2, where the rows of the matrix are the input snippets (objects), and the columns are the distinct words found in the input snippets (attributes), presented as 0 and 1.(0 - the distinct word not found in the input snippet, 1 - the distinct word found in the input snippet).

Since the rank-k in SVD is known to remove noise by ignoring small differences between row and column vectors of A (they will correspond to small singular values, which we drop by the choice of k), it can be used in our algorithm because SVD creates equivalence classes of data from the original data through deleting and adding some no primary attributes in the objects, this process leads the objects to have similarity in their attributes. From this similarity the algorithm combines these objects that have the same attributes and presents them like one object.

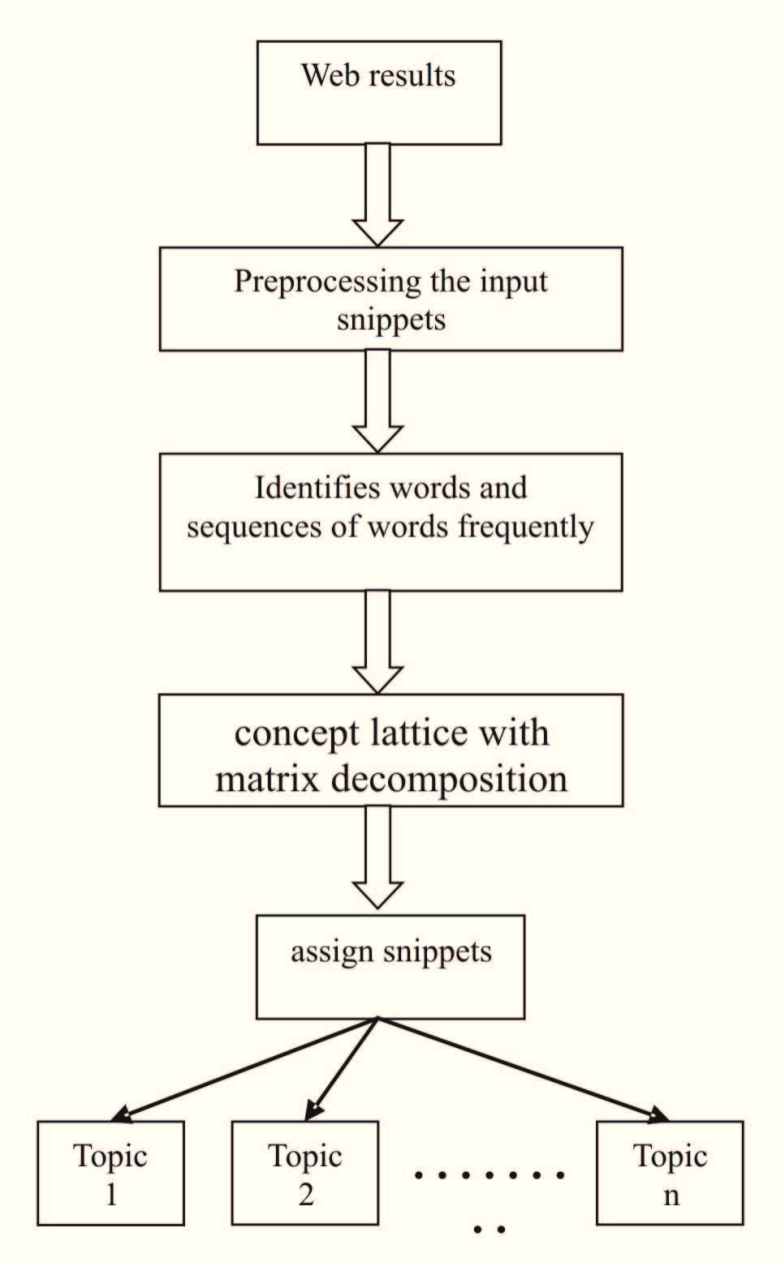


Figure 7.1: Steps of the algorithm process

7.2 Using Singular Value Decomposition SVD

I applied my experiment on data created from Google search engine contained from 1000 snippets with 20 distinct different words, but for easy explanation how the algorithm works, I deduct a small part from these data.

The new object created from this process represent a group of snippets has a similar distinct word. That means that the algorithm can minimize the huge volume of snippets received as a result from the searching in the web.

The data which I deducted from the result of Google search engine (figure 7.3) are 36 objects (snippets) with 9 attributes (distinct words). These data have peculiar combination in every object.

From the data in the table 7.3, I computed concept lattice to show the relation between the objects and there attributes. we can see the difficulty for reading this concept lattice (figure 7.1).

After using SVD with good choice k-rank, we can see that the output data from the SVD method are changed and brewed new attributes combinations in the objects. These combinations are repeated in more than one object (figure 7.4).

Some attributes are added and removed in some objects to be similar with another objects. Adding and deleting these attributes in objects brewed new attributes combinations in the objects, and these combination are repeated in more than one objects. From this repeating of attributes combination in the objects, that are not necessary to represent all the objects with the same attributes, we can only represent the objects that have peculiar combinations of attributes (figure 7.5).

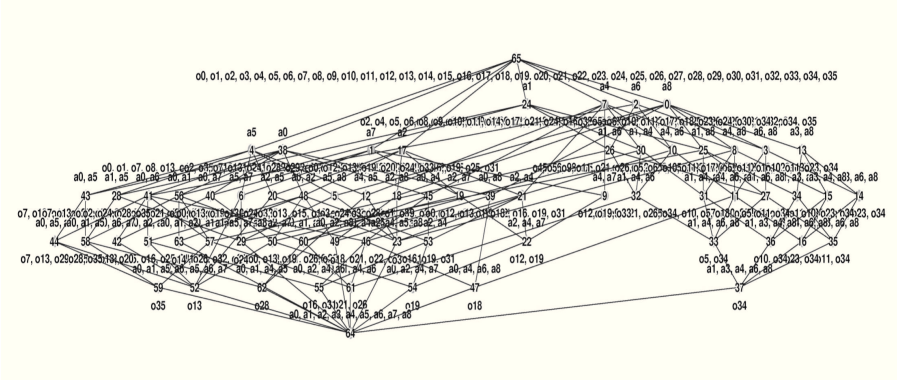


Figure 7.2: The concept lattice computed from the table 7.1

	A0	A1	A2	A3	A4	A5	A6	A7	A8
O0	1	0	1	0	0	0	0	1	0
O1	1	0	1	0	0	0	0	0	1
O2	0	1	0	0	0	1	0	0	0
O3	0	0	1	0	1	1	0	0	1
O4	0	1	0	0	0	0	1	0	0
O5	0	1	0	0	1	0	1	0	1
O6	0	1	0	0	1	0	0	0	1
O7	1	0	0	0	0	1	1	0	0
O8	1	1	0	0	1	0	0	0	0
O9	0	1	0	0	1	0	1	0	0
O10	0	1	0	1	1	0	0	0	1
O11	0	1	0	1	0	0	1	0	1
O12	0	0	1	0	1	0	0	1	0
O13	1	0	1	0	0	1	1	1	0
O14	1	1	1	0	0	0	0	0	0
O15	1	0	1	0	0	0	1	0	0
O16	1	0	1	0	1	0	1	0	0
O17	0	1	0	0	1	0	0	0	1
O18	1	0	0	0	1	0	1	0	1
O19	1	0	1	0	1	0	0	1	0
O20	1	0	0	0	0	0	1	1	0
O21	1	1	0	0	1	0	1	0	0
O22	1	0	0	0	1	0	1	0	0
O23	0	0	0	1	1	0	1	0	1
O24	0	1	0	0	0	1	0	1	1
O25	1	0	1	0	0	0	1	0	0
O26	1	1	0	0	1	0	1	0	0
O27	0	1	0	0	1	0	0	0	0
O28	1	1	0	0	1	1	0	0	0
O29	1	0	0	0	0	1	1	0	0
O30	0	1	0	0	1	0	0	0	1
O31	1	0	1	0	1	0	1	0	0
O32	1	1	0	0	0	0	1	0	0
O33	0	0	0	0	1	0	0	1	0
O34	0	1	0	1	1	0	1	0	1
O35	1	1	0	0	0	1	1	0	0

Figure 7.3: Data deducted from the result of Google search engine

	A0	A1	A2	A3	A4	A5	A6	A7	A8
O0	1	0	1	0	0	0	1	1	0
O1	0	0	1	0	0	0	1	0	0
O2	0	0	0	0	0	1	0	1	0
O3	0	0	1	0	0	0	0	0	0
O4	0	0	0	0	1	1	0	0	0
O5	0	0	1	0	0	0	1	0	0
O6	0	1	1	0	0	0	0	1	0
O7	0	0	0	0	1	1	0	0	0
O8	0	0	1	0	0	0	0	0	0
O9	0	0	1	0	0	0	0	0	0
O10	0	1	0	0	1	0	0	1	0
O11	0	1	0	0	1	0	0	0	0
O12	1	0	1	1	0	0	1	1	1
O13	1	0	1	0	0	0	1	1	0
O14	0	0	1	0	0	0	1	0	0
O15	1	0	1	0	0	0	1	1	0
O16	1	0	1	1	0	0	1	1	1
O17	0	0	0	0	0	1	0	0	0
O18	1	0	1	0	0	0	1	1	0
O19	1	0	1	1	0	0	1	1	1
O20	0	0	1	0	0	0	1	0	0
O21	0	0	1	0	1	0	0	0	0
O22	1	0	1	0	0	0	1	1	0
O23	0	0	1	0	0	0	1	1	0
O24	0	0	1	0	0	0	1	1	0
O25	1	0	1	0	0	0	1	1	0
O26	0	0	1	0	0	0	1	0	0
O27	0	1	0	0	1	0	0	0	0
O28	0	0	0	0	0	1	0	0	0
O29	0	0	1	0	0	0	1	0	0
O30	0	0	0	0	0	1	0	0	0
O31	1	0	1	1	0	0	1	1	1
O32	0	1	1	0	0	0	0	0	0
O33	0	0	1	0	0	0	0	0	0
O34	0	0	1	0	0	0	0	0	0
O35	0	1	1	0	0	0	0	0	0

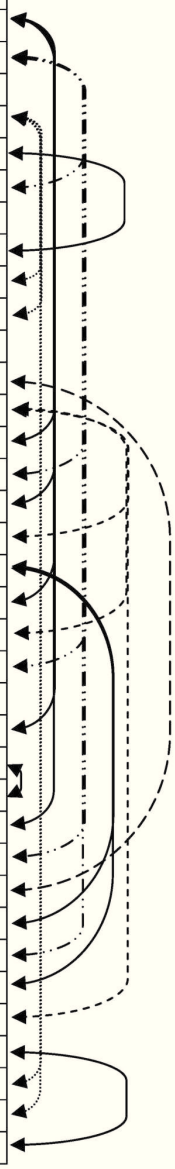


Figure 7.4: Data after using SVD with k-rank

This process minimizes the representing data to 36% from the original data with saving the primary attributes in the objects.

Minimizing the representing data after using SVD with k-rank, give us minimizing of view presenting the concept lattice which present those data.

After using the SVD the concept lattice is reduced and the number of nodes in the concept lattice after using SVD is less then the number of nodes in the concept lattice before using SVD. This decrease of nodes come from the combining of similar objects in the simplified formal context after using SVD (see figure 7.6).

	A0	A1	A2	A3	A4	A5	A6	A7	A8
A	0	0	0	0	0	1	0	0	0
B	0	0	0	0	0	1	0	1	0
C	0	0	0	0	1	1	0	0	0
D	0	0	1	0	0	0	0	0	0
E	0	0	1	0	0	0	1	0	0
F	0	0	1	0	0	0	1	1	0
G	0	0	1	0	1	0	0	0	0
H	0	1	0	0	1	0	0	0	0
I	0	1	0	0	1	0	0	1	0
J	0	1	1	0	0	0	0	0	0
K	0	1	1	0	0	0	0	1	0
L	1	0	1	0	0	0	1	1	0
M	1	0	1	1	0	0	1	1	1

Figure 7.5: Representing data after combining the similar Objects

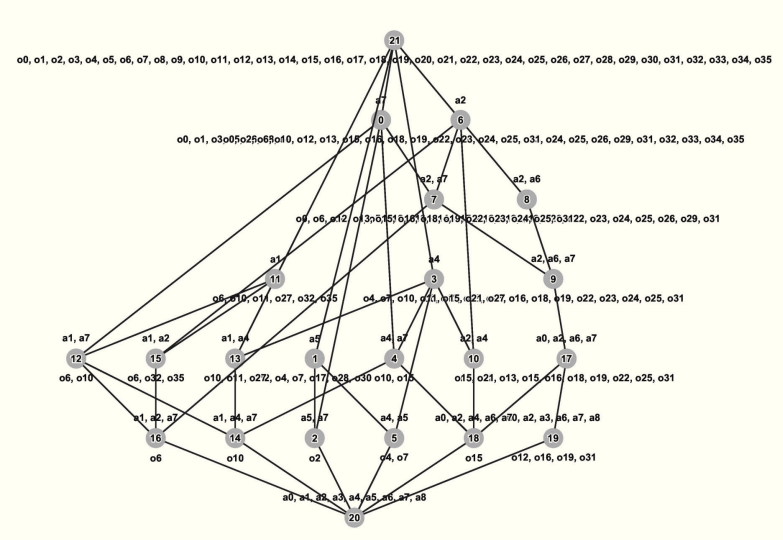


Figure 7.6: The concept lattice computed from the table 7.2

The activity of this method is shown clearly on the huge data with big quantity of attributes, and for a good choice k-rank from the diagonal matrix when applying SVD. Figure 7.7 shows how the k-rank value plays role in the number of attributes combination.

Figure 7.8 shows us the new combinations of objects and their content. We can see that, the 36 objects in the original data can showed like 13 groups after using SVD. For example group A content 3 objects (O17, O28, O30) which the attribute 5 is the common factor among them. And the group E content 6 objects (O1, O5, O14, O20, O26, O29) which the attributes 2 and 6 are the common factor

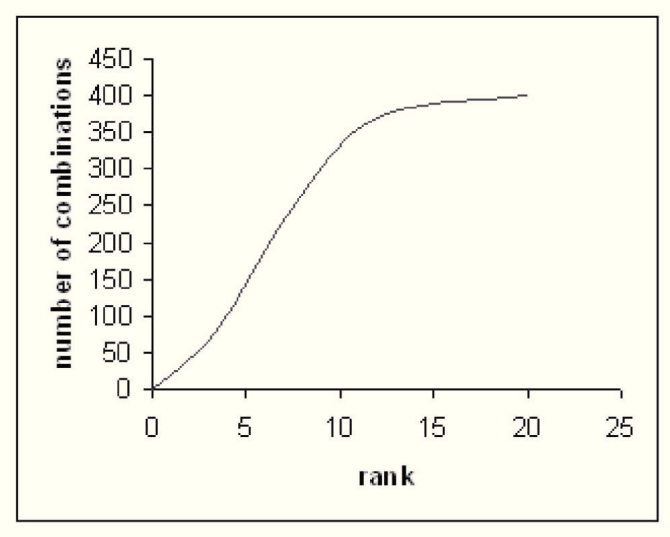


Figure 7.7: Relationship between value of k-rank and number of combination

among them.

Groups	Number of objects	Objects	Attributes
A	3	O17, O28, O30	A5
B	1	O2	A5, A7
C	2	O4, O7	A4, A5
D	5	O3, O8, O9, O33, O34	A2
E	6	O1, O5, O14, O20, O26, O29	A2, A6
F	2	O23, O24	A2, A6, A7
G	1	O21	A2, A4
H	2	O11, O27	A1, A4
I	1	O10	A1, A4, A7
J	2	O32, O35	A1, A2
K	1	O6	A1, A2, A7
L	6	O0, O13, O15, O18, O22, O25	A0, A2, A4, A6, A7
M	4	O12, O16, O19, O31	A0, A2, A3, A6, A7, A8

Figure 7.8: content of the groups

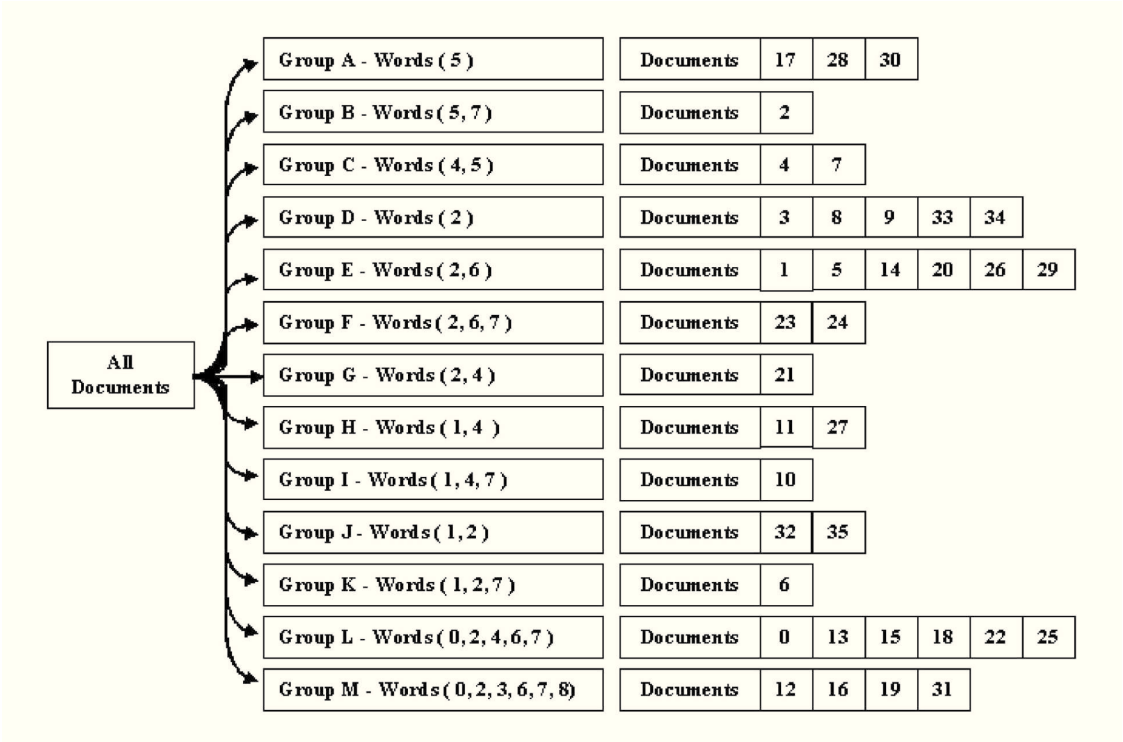


Figure 7.9: Map of the groups and their content

8 Conclusions

Using concept lattice for visualizing has negative points, the dimension of the computed lattices very often do not fit in the screen even for small input data. Many researchers have developed interfaces that allow multiple local and global views, but these methods consume time and capacity.

In this work I deal with the possible usage of matrix decompositions to improve concept lattice reduction where there are some goals of this work. These goals compose the main ideas to be solved by this thesis. One of these ideas has to do with minimizing the input data before constructing the concept lattice. This work also aims at using the matrix decompositions to improve concept lattice reduction, as well as, using two methods of matrix decomposition, Singular Value Decomposition (SVD) and Non-Negative Matrix Factorization (NMF). Another goal set for this work is to show the different behavior of the concept lattice reduction before and after using methods of matrix decompositions. One of the important aims set by this research is to use matrix decomposition and concept lattice in search result clustering. Eventually, presenting and describing the different uses of matrix decomposition and concept lattice as a mathematical method to reduce a big value of objects in search result clustering by combining the attributes of these objects.

Concept lattice is computed from context data which is represented by the relationship between the objects and their attributes. When the objects does not have a similarity in their attributes, drawing the relationship edge between the nodes is very difficult and complicated.

The difficulty lies in the overlapping lines on each other, and this causes blurred vision in the normal size of the screen. For that reason, I proposed to minimize the context data before computing the concept lattice. To minimize the context data before computing the concept lattice, I used the matrix decomposition as a mathematical method to transform the data in a way that exposes the amount of variation in the data relative to a set of latent features. In this case, I used two methods of matrix decomposition: singular value decomposition SVD and Non-negative Matrix Factorization (NMF).

Every method has a different way to minimize data, the singular value decomposition (SVD) adds and deletes some non-essential attributes from the objects. This adding and deleting of the attributes leads to attributes' similarity in some objects. This similarity gives us matching in some objects that have a similarity in their attributes.

On the other hand, the Non-negative Matrix Factorization (NMF) adds some non-essential attributes to the objects. Adding of these attributes gives us attributes similarity in some objects. This similarity gives us matching in some

objects that have a similarity in their attributes.

As a result, this matching in the objects do not necessarily represent all the objects, but represents only non similar objects. Thus, every object will represent a group of objects. This means the objects will cluster into some groups.

When calculating the concept lattice from these clustered objects in groups, the concept lattice will be minimized and clearer.

A few steps were executed to perform this experiment as explained below.

The first step is to prepare the data, which will be the input data for this experiment. The format of this data is binary matrix. The content of this matrix is 1 and 0, when 1 means that the object has the attribute and 0 means that the object has no attribute.

The second step is to compute the concept lattice from the input data, to see the difficulty of understanding the relationship between the nodes representing the objects. This difficulty of understanding the relationship between the nodes comes because of the lack of clarity of concept lattice.

The third step is to pass the input data to the software of singular value decomposition (SVD) to cluster these data. This step changes the original data to be clustered in groups because of the deletion and addition of some non-essential attributes in some objects.

The fourth step is to compute the concept lattice from the data after using singular value decomposition (SVD), and analyze the content of the nodes in the concept lattice. This step shows the difference between the original lattice and the lattice after using singular value decomposition (SVD). We can see that the lattice after using singular value decomposition (SVD) has fewer nodes than the original lattice. This decrease in the nodes comes as a result of the clustering data done by singular value decomposition (SVD).

The third and fourth steps are proved in the following papers authored by me: [1][3][4][5][7][13].

The fifth step is to pass the original input data to the software of Non-negative Matrix Factorization (NMF) to cluster these data. This step changes the original data to be clustered in groups as a result of the addition of some non-essential attributes in some objects.

The sixth step is to compute the concept lattice from the data after using Non-negative Matrix Factorization (NMF), and analyze the content of the nodes in the concept lattice. This step shows us the difference between the original lattice and the lattice after using Non-negative Matrix Factorization (NMF). We can see that the lattice after using Non-negative Matrix Factorization (NMF) has less nodes then the original lattice, this decrease in the nodes comes as a result of the clustering data done by Non-negative Matrix Factorization (NMF).

The fifth and sixth steps are proved in the following papers authored by me [5][7][8][11].

The seventh step is to study the behavior of concept lattice after using both methods of matrix decomposition (singular value decomposition (SVD) and Non-

negative Matrix Factorization (NMF).

The seventh step is proved in the following paper authored by me [2]

This experiment has been proved highly successful, and the objectives of this work has been achieved.

After the success of the experiment, I have applied it on search result clustering because of its importance and because of the large number of data resulting from it. We can use the concept lattice to represent the document resulting from the search Engine and use the matrix decomposition to improve the concept lattice reduction.

For this experiment, I used real data taken from Google search engine and applied the steps of the previous experiment on these data.

This experiment shows us clearly the success of this method to improve the concept lattice as a tools in search result clustering.

This experiment is proved in the following papers authored by me [6][9][10].

This work deals with possible usage of matrix decompositions to improve concept lattice reduction. For the purpose of this study, this work uses two methods for matrix decomposition, singular value decomposition (SVD) and Non-negative Matrix Factorization (NMF).

In this thesis, I applied these methods in the area of formal concept analysis (FCA). After applying the above-mentioned methods, the aim of this study, which has to do with minimizing the input data before constructing the concept lattice, was attained.

This study has come to a few other conclusions, one of which has to do with the reduction of the input data. The reduction of the input data has lower dimension than the original data. This leads to another conclusion which is that computing the lattice leads in creating smaller graphic representation in less time, which was also the aim of this thesis.

Another major conclusion that this study has arrived to is that this work has presented the difference in the behavior of the concept lattice reduction after using various methods of matrix decompositions.

In addition to that, this study has come to the conclusion that by applying these methods on Search Result Clustering, the results proved to be very positive and proved the feasibility of the methods used.

8.1 Future work

As was mentioned above, the area of using concept lattice and matrix decomposition for clustering and viewing data is a very important topic, because the amount of data that we get from technological advances is growing every day.

Trying different other types of matrix decomposition unused in this work will give us a wider field of research and may enable us to obtain outstanding results so to enable us to improve this method.

Recently, I am working on other methods such as, Semi Discrete matrix decomposition SDD, Eigen matrix decomposition, Hessenberg matrix decomposition and Jordan matrix decomposition.

The method I used in this thesis can be applicable on process-structured data, where each part has different contents. An example of these data is the data applicable on emails. Contents of the emails have huge amount of topics, and clustering these emails by concept lattice will be very useful. This will be one of the important future topics.

9 Author's publications

Summarization:

IEEE Xplore (http://ieeexplore.ieee.org)	12
ISI (Web of Science, http://apps.webofknowledge.com)	9
Scopus (http://www.scopus.com/)	9
DBLP (http://dblp.uni-trier.de/db/)	8
Springerlink (http://link.springer.com)	1
Other	1

[1] V. Snasel, P. Gajdos, H. D. Abdulla & M. Polovincak. Concept Lattice Reduction by Matrix Decompositions. first International Conference on Digital Communications and Computer Applications, DCCA 2007. 35-42. Amman, Jordon , 2007.

[2] V. Snasel, H. D. Abdulla & M. Polovincak. Behavior of the Concept Lattice Reduction to visualizing data after Using Matrix Decompositions. IEEE Innovations'07.392-396. Dubai, UAE, 2007. (IEEE Xplore, ISI, Scopus)

[3] V. Snasel, M. Polovincak & H. D. Abdulla. Concept Lattice Reduction by Singular Value Decomposition. SYRCoDIS 2007, Moscow, Russian, 2007. (DBLP)

[4] V. Snasel, P. Gajdos, H. D. Abdulla & M. Polovincak. Using Matrix Decompositions in Formal Concept Analysis. 10th International Conference on Information System Implementation and Modeling ISIM 2007.121-128. OPAVA, Czech Republic, 2007. (DBLP)

[5] V Snasel, M. Polovincak, H. D. Abdulla & Z. Horak. On Knowledge Structures Reduction. Computer Information Systems and Industrial Management Applications, CISIM '08. 7th IEEE CISIM.33-37. Ostrava, Czech Republic, 2008. (IEEE Xplore, ISI, DBLP)

[6] V. Snasel, & H. Abdulla. Solution for serch result clustring by using Singular Value Decomposition. International Conference on Advanced Computer Theory and Engineering, IEEE ICACTE '08. 647 - 651, Phuket, Thailand.2008. (IEEE Xplore)

[7] V. Snasel, M. Polovincak, H. D. Abdulla & Z. Horak. On Concept Lattices and Implication Bases from Reduced Contexts. International Conference on Conceptual Structures (ICCS).83-90. Toulouse, France, 2008. (DBLP)

[8] V. Snasel, H. D. Abdulla & M. Polovincak. Using Nonnegative Matrix Factorization and Concept Lattice Reduction to visualizing data. First International Conference on Applications of Digital Information and Web Technologies, IEEE ICADIWT. 296-301. Ostrava, Czech Republic, 2008. (IEEE Xplore, ISI, Scopus)

[9] H. D. Abdulla & V. Snasel. Using Singular Value Decomposition (SVD) as a solution for search result clustering. IEEE Innovations'08. 302-306. Al Ain, UAE, 2008. (IEEE Xplore, ISI)

[10] V. Snasel & H. D. Abdulla. Search Result Clustering using a Singular Value Decomposition. In Proceedings of the First International Conference on Intelligent Human Computer Interaction. Springer, Part 4, 336-343, Allahabad, India, 2009. (Springerlink)

[11] V. Snasel, H. D. Abdulla & M. Polovincak. Clustering data using a Nonnegative Matrix Factorization (NMF). In Applications of Digital Information and Web Technologies 2009. ICADIWT '09. Second International Conference . IEEE Computer Society. 749-752. Los Alamitos, USA, 2009. (IEEE Xplore, ISI)

[12] V. Snasel, S.A.R.S. Al-Dubaee, H. D. Abdulla & N. Ahmad. A New Search Result Clustering Using Haar Wavelet Transform.International Conference on Future Computer and Communication, IEEE ICFCC 2009. 653-657. Aligarh, India. 2009. (IEEE Xplore, ISI, Scopus)

[13] H. D. Abdulla, M. Polovincak & V. Snasel. Using a Matrix Decomposition for Clustering Data. IEEE International Conference on Computational Aspects of Social Networks CASoN 2009.24-27. Fontainebleau, France. 2009. (IEEE Xplore, DBLP, ISI, Scopus)

[14] J. Wu, H. D. Abdulla & V. Snasel. Application of Binocular Vision and Diamond Search Technology in Computer Vision Navigation. IEEE International Conference on Intelligent Networking and Collaborative Systems, INCOS '09. 87-92. Fontainebleau, France, 2009. (IEEE Xplore, DBLP, ISI, Scopus)

[15] V. Snasel, J. Wu & H. D. Abdulla. A Novel Approach of Computer Vision Navigation for Mobile Tracking Robot. In Computational Intelligence, Modelling and Simulation, CSSim '09. IEEE Computer Society. 36-42. Los Alamitos, USA, 2009. (IEEE Xplore, Scopus)

[16] A. M. El Tahir Ali, H. D. Abdulla & V. Snasel. Using Kohonen Maps and Singular Value Decomposition for Plagiarism Detection. Third International

Conference on Computational Intelligence, Communication Systems and Networks (CICSYN). 60-64, Bali, Indonesia, 2011. (IEEE Xplore, DBLP, Scopus)

[17] A. M. El Tahir Ali, Hussam M. D. Abdulla & V. Snasel. Survey of Plagiarism Detection Methods. IEEE Fifth Asia Modelling Symposium. 39-42. Manila, Philippines, 2011. (IEEE Xplore, Scopus)

[18] A. M. El Tahir Ali, Hussam M. D. Abdulla & V. Snasel. Overview and Comparison of Plagiarism Detection Tools. Proceedings of the Dateso 2011: Annual International Workshop on Databases, Texts, Specifications and Objects. 161-172. Pisek, Czech Republic, 2011. (DBLP)

[19] H. Soori, J. Platos, V. Snasel & H. Abdulla. Simple rules for syllabification of arabic texts. International Conference on Digital Information Processing and Communications, ICDIPC 2011. Volume 188 CCIS, Issue PART 1. 97-105. Ostrava, Czech Republic, 2011. (ISI, Scopus)

10 References

[1] Y. Kalfoglou, S. Dasmahapatra, & Y. Chen-Burger. Fca in knowledge technologies: Experiences and opportunities. In Proceedings of 2nd International Conference on Formal Concept Analysis, 252-260. Springer. 2004.

[2] C. Carpineto & G. Romano. A lattice conceptual clustering system and its application to browsing retrieval. Machine Learning 24, 95-122. 1996.

[3] C. Carpineto & G. Romano. Concept Data Analysis: Theory and Applications. Wiley. 2004.

[4] T. J. Everts, S. S. Park & B. H. Kang. Using formal concept analysis with an incremental knowledge acquisition system for web document management. In Proceedings of the 29th Australasian Computer Science Conference, 247-256. 2006.

[5] G. Snelting & F. Tip. Reengineering class hierarchies using concept analysis. In Proceedings of the 6th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 99-110. 1998.

[6] V. Choi, Y. Huang, V. Lam, D. Potter, R. Laubenbacher & K. Duca. Using formal concept analysis for microarray data comparison. In Proceedings of the 5th Asia Pacific Bioinformatics Conference, 57-66. 2006.

[7] C. Caprineto & G. Romano. GALOIS: A lattice conceptual clustering system and its application to browsing retrieval. Proceedings of the Tenth International Conference on Machine Learning, MA: Morgan Kaufmann, 95-122. Amherst, USA. 1993.

[8] R. Cole, P. Eklund & G. Stumme. CEM - a program for visualization and discovery in email. In 4th European conference on principles and practice of knowledge discovery in databases, 367-374, Springer. 2000.

[9] R. Godin, R. Missaoui & H. Alaoui. Learning Algorithms Using a Galois Lattice Structure. Proceedings of the Third International Conference on Tools for Artificial Intelligence. IEEE Computer Society Press, 22-29. 1991.

[10] D. Cutting, D. Karger, J. Pedersen, & J. Tukey. Scatter/Gather: A cluster-based approach to browsing large document collections. Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 318-329. USA. 1992.

[11] R. Wille. Concept Lattices and Conceptual Knowledge Systems. Computers and Mathematics with Applications 23, 493-515. 1992.

[12] M. Berry & M. Brown. Understanding Search Engines. Mathematic al Modeling and Text Retrieval, Siam, 1999.

[13] M. Berry, S. Dumais & T. Letsche. Computation Methods for Intelligent Information Access. Proceedings of the 1995 ACM/IEEE Supercomputing Conference. 1995.

[14] B. Ganter & R. Wille. Formal Concept Analysis. Springer Verlag. Berlin. 1996.

[15] Lin X. Map displays for information retrieval. Journal of the American Society of Information Science, 40-54. 1997.

[16] P. Bruza & S. Dennis. Query reformulation on the internet: empirical data and the hyper index search engine. Proceedings of the RIAO97 Conference-Computer-Assisted Information Searching on Internet. 1997.

[17] P. Bruza & R. McArthur. Interactive internet search: Keyword, directory and query reformulation mechanisms compared. Proceedings of the 23rd Annual ACM Conference of Research and Development in Information Retrieval. SIGIR. 2000.

[18] R. McArthur & P. Bruza. The ranking of query refinements in interactive web-based retrieval. Proceedings of the Information Doors Workshop (Held in conjunction with the ACM Hypertext and Digital Libraries Conferences). 2000.

[19] G. Birkhoff. Lattice Theory. American Mathematical Society, Third Edition. USA. 1995.

[20] P. Crawley & R. P. Dilworth. Algebraic Theory of Lattices. Prentice Hall. 1973.

[21] B. Davey & H. riestley. Introduction to Lattice and Order. Cambridge University Press. 1990.

[22] R. Wille. Lattices in data analysis: How to draw them with a computer, Algorithms and Order. 1989.

[23] B. Ganter & G. Stumme. Formal Concept Analysis: Methods and Applications in Computer Science. 2002.

[24] J. Sowa. Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley. 1984.

[25] R. Wille. Restructuring Lattice Theory: an aroach based on hierarchies of concepts. Ordered Sets. 1982.

[26] R. Godin, R. Missaoui & H. Alaoui. Incremental concept formation algorithms based on Galois (concept) lattices. Computational Intelligence. 246-267. 1995.

[27] G. D. Oosthuizen. The application of concept lattices to machine learning. Technical Report CSTR 94/01, University of Pretoria. 1994.

[28] S. O. Kuznetsov & S. A. Obiedkov. Algorithms for the Construction of Concept Lattices and Their Diagram Graphs. Lecture Notes in Computer Science. 2001.

[29] D. E. Knut. The Art of Computer Programming, Volume 3, Sorting and Searching. Addison-Wesley. 1998.

[30] R. Wille. Lattices in data analysis: how to draw them with a computer. Algorithms and order. Kluwer. 1989.

[31] G. Stumme & R. Wille. Ageometrical heuristic for drawing concept lattices. Graph Drawing. Springer. 1995.

[32] L. Hubert, J. Meulman & W. Heiser. Two purposes for matrix factorization: A historical appraisal. 2000.

[33] G. H. Golub & C. F. v. Loan. Matrix Computations. Johns Hopkins University Press, 3rd edition. 1996.

[34] T. G. Kolda & D. P. OLeary. Computation and uses of the semidiscrete matrix decomposition. ACM Transactions on Information Processing, 415-435. 1999.

[35] T. G. Kolda & D. P. OLeary. Latent Semantic Indexing Via A SemiDiscrete Matrix Decomposition, volume 107 of IMA Volumes in Mathematics and Its Applications. Springer Verlag. 1999.

[36] A. Hyvarinen & E. Oja. Independent component analysis: Algorithms and applications. Neural Networks, 2000.

[37] D. Skillicorn. Understanding Complex datasets: data mining with matrix decompositions. Data Mining and Knowledge Discovery Series. 2007.

[38] M. Berry & M. Browne. Understanding Search Engines, Mathematical Modelling and Text Retrieval. Siam. 1999.

[39] M. Berry, S. Dumais & T. Letsche. Computation Methods for Intelligent Information Access, In Proceedings of the 1995 ACM/IEEE Supercomputing Conference. 1995.

[40] R. M. Larsen. Lanczos bidiagonalization with partial reorthogonalization. Technical report, University of Aarhus. 1998.

[41] D. Kalman. A singularly valuable decomposition: The SVD of a matrix. College Math Journal. 1996.

[42] G. Young & C. Eckart. The approximation of one matrix by another of lower rank. Psychometrika 1, 211-218. 1936.

[43] G. H. Golub & C. F. v. Loan. Matrix Computations. Johns Hopkins University Press. 1996.

[44] G. W. Stewart. On the early history of the Singular Value Decomposition. Technical Report TR-2855, University of Maryland, Department of Computer Science. 1992.

[45] A. Karol & M. A. Williams. Understanding human strategies for change: an empirical study. In TARK '05: Proceedings of the 10th conference on Theoretical aspects of rationality and knowledge, 137-149. National University of Singapore, 2005.

[46] L. Hubert, J. Meulman & W. Heiser. Two purposes for matrix factorization: A historical appraisal, 68-82. SIAM Review, 2000.

[47] A. Kontostathis & W. M. Pottenger. Detecting patterns in the LSI term-term matrix. Technical Report LU-CSE-02-010, Department of Computer Science and Engineering, Lehigh University. 2002.

[48] A. Kontostathis & W. M. Pottenger. Improving retrieval performance with positive and negative equivalence classes of terms. Technical Report LU-CSE-02-009, Department of Computer Science and Engineering, Lehigh University. 2002.

[49] O. Alter, P. O. Brown & D. Botstein. Singular value decomposition for genome-wide expression data processing and modeling. Proceedings of the National Academy of Science, 10101-10106. USA. 2000.

[50] M. Zhu & A. Ghodsi. Automatic dimensionality selection from the scree plot via the use of profile likelihood. Computational Statistics and Data Analysis, 918-930. 2006.

[51] D. Achlioptas & F. McSherry. Fast computation of low rank matrix approximations. In Journal of the ACM Symposium on Theory of Computing (STOC). 2001.

[52] M. T. Chu. On the statistical meaning of truncated singular value decomposition. Preprint. 2001.

[53] C. H. Q. Ding, X. He & H. Zha. A spectral method to separate disconnected and nearly-disconnected Web graph components. Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data miningIn KDD01, 275-280. New York, USA. 2001.

[54] C. J. Alpert & S. Z. Yao. Spectral partitioning: The more eigenvectors, the better. In 32nd ACM/IEEE Design Automation Conference, 195-200. SAN FRANCISCO, USA, 1995.

[55] P. Castro. Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation. An Official Journal of the American Society for Quality and the American Statistical Association, 132-133. 1980.

[56] D. B. Skillicorn & X. Yang. High-performance singular value decomposition. Data mining for scientific and engineering applications, 401-424. Minneapolis, USA. 2001.

[57] D. Lee & H. Seung. Learning of the parts of objects by non-negative matrix factorization. Nature 401, 788-791. 1999.

[58] Y. Cho & S. Choi. Nonnegative features of spectro-temporal sounds for classification. Pattern Recognition Letters 26, 1327-1336. 2005.

[59] P. Sajda, S. Du & L. Parra. Recovery of constituent spectra using non-negative matrix factorization. Proceedings of the society of photo-optical instrumentation engineers (spie), 321-331. San Diego, USA. 2003.

[60] D. Guillamet, J. Vitria & B. Schiele. Introducing a weighted nonnegative matrix factorization for image classification. Pattern Recognition Letters 24, 24472454. 2004.

[61] H. Li, T. Adali & D. Wang. Non-negative matrix factorization with orthogonally constraints for chemical agent detection in Raman spectra. IEEE Workshop on Machine Learning for Signal Processing, 253-258. Mystic, USA. 2005.

[62] A. Cichocki, R. Zdunek & S. Amari. Csiszar's divergences for non-negative matrix factorization. Proceedings of the 6th International Conference on Independent Component Analysis and Blind Signal Separation, 32-39. Charleston, USA. 2006.

[63] P. Paatero & U. Tapper. Positive matrix factorization: A nonnegative factor model with optimal utilization of error estimates of data values. Environmetrics 5, 111126. 1994.

[64] E. Oja & M. Plumbley. Blind separation of positive sources using nonnegative PCA. 4th International Symposium on Independent Component Analysis and Blind Signal Separation. Japan. 2003.

[65] P. Hoyer. Non-negative matrix factorization with sparseness constraints. Journal of Machine Learning Research 5, 1457-1469. 2004.

[66] R. Kompass. A generalized divergence measure for nonnegative matrix factorization. Neural Computation 19, 780-791. 2007.

[67] D. Lee & H. Seung. Algorithms for non-negative matrix factorization. NIPS, Neural Information Processing Systems. 2000.

[68] V. Pauca, F. Shahnaz, M. Berry & R. Plemmons. Text mining using non-negative matrix factorizations. Proceedings of the fourth SIAM international conference on data mining SIAM, 452-456. Lake Buena Vista, FL. 2004.

[69] H. Zeng, Q. He, Z. Chen, W. Ma & J. Ma. Learning to cluster web search results. Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval, 210-217. New York, USA. 2004.

[70] G. Salton. Automatic Information Organization and Retrieval. McGraw-Hill. New York, USA. 1968.

[71] M. Barbut & B. Monjardet. Classification and order. Algebra and Combinatorics. Tome II. Paris. 1970.

[72] S. M. Gusakova & S. O. Kuznetsov. Foundations and Applications of Conceptual Structures Contributions. Bulgarian Academy of Sciences. 2002.

[73] H. Dyvik. A Translational Basis for Semantics. In H. Johansson, & S. Oksefjell (Eds.), Corpora and Crosslinguistic Research: Theory, Method and Case Studies, 51-57. Rodopi. 1998.

[74] J. Barwise & J. Seligman. Information Flow. The Logic of Distributed Systems. Cambridge University Press. 1997.

[75] E. Rosch. Natural categories. Cognitive Psychology 4, 328350. 1973.

[76] D. L. Medin. Concepts and Conceptual Structure. American Psychologist 12, 1469-1481. 1989

[77] U. Priss. Relational Concept Analysis: Semantic Structures in Dictionaries and Lexical Databases. Aachen, Shaker. 1998.

[78] U. Priss. Lattice-based Information Retrieval. Knowledge Organization, 27, 3. 2000.

[79] R. Godin, J. Gecsei & C. Pichet. Design of browsing interface for information retrieval. SIGIR '89 Proceedings of the 12th annual international ACM SIGIR conference on Research and development in information retrieval, 32-39. New York, USA. 1989.

[80] R. Godin, R. Missaoui & A. April. Experimental comparison of navigation in a Galois lattice with conventional information retrieval methods. International journal of man-machine studies 38, 747-767. 1993.

[81] H. Mili, E. Ah-Ki, R. Godin & H. Mcheick. Another nail to the coffin of faceted controlled-vocabulary component classification and retrieval. Proceedings of the 1997 symposium on Software reusability SSR '97, 89-98. New York, USA. 1997.

[82] C. Carpineto & G. Romano. GALOIS: An order-theoretic approach to conceptual clustering. Proceedings of the 10th conference on Machine Learning. Amherst, MA, Kaufmann. 1993.

[83] C. Carpineto & G. Romano. Exploiting the Potential of Concept Lattices for Information Retrieval with CREDO. Journal of Universal Computing 10, 985-1013. 2004.

[84] C. Carpineto & G. Romano. Information retrieval through hybrid navigation of lattice representations. International Journal of Human-Computer Studies, 45. 1996.

[85] C. Carpineto & G. Romano. A lattice conceptual clustering system and its application to browsing retrieval. Machine Learning 24, 95-122. 1996.

[86] M. Skorsky. Graphical representation of a thesaurus. German Documentalists. Regensburg. 1997.

[87] U. Priss. A Graphical Interface for Document Retrieval Based on Formal Concept Analysis. Proceedings of the 8th Midwest Artificial Intelligence and Cognitive Science Conference. AAAI Technical Report CF-97-01. 1997.

[88] P. Eklund, J. Ducrou & P. Brawn. Concept Lattices for Information Visualization: Can Novices Read Line Diagrams?. Second International Conference on Formal Concept Analysis. Concept Lattices, Lecture Notes in Computer Science, 235-236. Springer. Berlin. 2004.

[89] R. Cole & P. Eklund. Browsing Semi-structured Web Texts Using Formal Concept Analysis. Conceptual Structures: Broadening the Base, 319-332. Springer. Berlin. 2001.

[90] R. Cole & G. Stumme. CEM - A Conceptual Email Manager. Conceptual Structures: Logical, Linguistic and Computational Issues, 438-452. Springer. Berlin. 2000.

[91] S. Dumais. Latent semantic analysis. Annual Review of Information Science and Technology (ARIST) 38, 188230. 2004.

[92] T. Rock & R. Wille. A TOSCANA - Sensing system for literature search. Conceptual Knowledge Processing. Methods and Applications. Springer. Berlin. 2000.

[93] R. Wille. Restructuring lattice theory: an approach based on hierarchies of concepts. Formal Concept Analysis: 7th International Conference. Darmstadt. Germany. 1982.

[94] V. Hentig. Mage or Master? On the Unity of Science in the amplification andi manufacturing process. Stuttgart. 1972.

[95] R. Wille. Conceptual landscapes of knowledge: a pragmatic paradigm for knowledge processing. Classification in the Information Age. Springer. Berlin. 1999.

[96] M. Scaife & Y. Rogers. External cognition: how do graphical representations work. International Journal of Human-Computer Studies 45. 1996.

[97] F. Dau. Types and Tokens for Logic with Diagrams. Conceptual Structures at Work. 12th International Conference on Conceptual Structures. Springer. Berlin. 2004.

[98] G. Stumme. Formal Concept Analysis on Its Way from Mathematics to Computer Science. Conceptual Structures: Integration and Interfaces, 10th International Conference on Conceptual Structures, 2-19. Springer. Berlin. 2002.

[99] R. Davis, H. Shrobe & P. Szolovits. What is a knowledge representation?. AI Magazine 14, 17-34. 1993.

[100] J. Hereth, G. Stumme, R. Wille & U. Wille. Conceptual Knowledge Discovery and Data Analysis. Conceptual Structures: Logical, Linguistic and Computational Issues, 421-437. Springer. Berlin. 2000.

[101] G. Stumme, R. Wille & U. Wille. Conceptual knowledge discovery in databases using formal concept analysis methods. Principles of Data Mining and Knowledge Discovery, 450-458. Springer. Berlin. 1998.

[102] W. Kollewe, M. Skorsky, F. Vogt & R. Wille. TOSCANA - a tool for conceptual analysis and exploration of data. Conceptual Knowledge Processing - Basic questions and tasks. Mannheim, B.I. Wissenschaftsverlag. 1994.

[103] R. Wille. Why Can Concept Lattices Support Knowledge Discovery in Databases?. Journal of Experimental & Theoretical Artificial Intelligence 14, 81-92. 2002.

[104] S. Kuznetsov & S. Obiedkov. Comparing Performance of Algorithms for Generating Concept Lattices. Journal of Experimental & Theoretical Artificial Intelligence 14, 189-216. 2002.

[105] S. Prediger & G. Stumme. Theory-driven Logical Scaling. Conceptual Information Systems meet Description Logics. Proceedings DL'99. CEUR Workshop Proc. 22. 1999.

[106] J. Sowa. Conceptual Structures: Information Processing in Mind and Machine. Addison-Wesley. 1984.

[107] R. Wille. Conceptual Graphs and Formal Concept Analysis. Conceptual Structures: Fulfilling Peirce's Dream, 290-303. Springer. Berlin. 1997.

[108] G. Mineau, G. Stumme & R. Wille. Conceptual Structures Represented by Conceptual Graphs and Formal Concept Analysis. Conceptual Structures: Standards and Practices, 423-441. Springer. Berlin. 1999.

[109] R. Wille. Contextual logic summary. Working with Conceptual Structures. Contributions to ICCS 2000. Aachen, Shaker. 2000.

[110] S. Prediger. Contextual judgment logic with term graphs. A contribution to the restructuring of mathematical logic. Aachen. Shaker. 1998.

[111] R. Wille. Boolean Concept Logic. Conceptual Structures: Logical, Linguistic and Computational Issues, 317-331. Springer. Berlin. 2000.

[112] R. Wille. Preconcept Algebras and Generalized Double Boolean Algebras. Concept Lattices: Second International Conference on Formal Concept Analysis, 237-238. Springer. Berlin. 2004.

[113] B. Ganter & R. Wille. Formal Concept Analysis. Mathematical Foundations. Springer. Berlin. 1999.

[114] F. Dau. Negations in Simple Concept Graphs. Conceptual Structures: Logical, Linguistic and Computational Issues, 263-276. Springer. Berlin. 2000.

[115] L. Kwuida, A. Tepavcevic & B. Seselja. Negation in Contextual Logic. Conceptual Structures at Work: 12th International Conference on Conceptual Structures, 234-235. Springer. Berlin. 2004.

[116] P. Burmeister & R. Holzer. On the Treatment of Incomplete Knowledge in Formal Concept Analysis. Conceptual Structures: Logical, Linguistic and Computational Issues, 385-398. Springer. Berlin. 2000.

[117] J. Hereth Correia & J. Klinger. Protoconcept Graphs: The Lattice of Conceptual Contents. Concept Lattices: Second International Conference on Formal Concept Analysis, 233-234. Springer. Berlin. 2004.

[118] R. Wille. Existential Concept Graphs of Power Context Families. Conceptual Structures Integration and Interfaces, 382-395. Springer. Berlin. 2002.

[119] Matlab, The Language of Technical Computing.

http://www.mathworks.com/products/matlab/. Last access 28.8.2012

[120] GNU Octave. http://www.gnu.org/software/octave/. Last access 28.08.2012.

[121] B. Ganter. Two basic algorithms in concept analysis. Formal Concept Analysis, volume 5986 of Lecture Notes in Computer Science, 312-340. Springer. Heidelberg, Berlin. 2010.

[122] U. Priss. Formal concept analysis in information science. Annual Review of Information Science and Technology 40, 521-543.2006.

[123] R. Belohlavek & V. Vychodil. What is a fuzzy concept lattice?. In CLA2005, Proceedings of the 3rd International Workshop 162, 34-45. Olomouc, Czech Republic. 2005.

[124] P. Eklund, J. Ducrou & P. Brawn. Concept lattices for information visualization: Can novices read line-diagrams?. In Concept Lattices, volume 2961 of Lecture Notes in Computer Science, 235-236. Springer Berlin. 2004.

[125] U. Priss. Linguistic applications of formal concept analysis. Formal Concept Analysis, 149-160. 2005.

[126] D. Poshyvanyk & A. Marcus. Combining formal concept analysis with information retrieval for concept location in source code. In Proceedings of the 15th IEEE International Conference on Program Comprehension, ICPC'07, 37-48. Washington DC, USA. 2007.

[127] E. Loisant, J. Martinez, H. Ishikawa & K. Katayama. Galois' lattices as a classication technique for image retrieval. Information and Media Technologies 1, 994-1006. 2006.

[128] M. Klimushkin, D. Chetverikov & A. Novokreshchenova. Formal concept analysis of the us blogosphere during the 2008 presidential campaign. 2008.

[129] P. du Boucher-Ryan & D. G. Bridge. Collaborative recommending using formal concept analysis. knowledge based systems19, 309-315. 2006.